Olli Niinioja

Lokalisointi- ja valuuttaparituen toteutus PlanMill-toiminnanohjausjärjestelmään

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Insinöörityö
6.6.2012

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Olli Niinioja Lokalisointi- ja valuuttaparituen toteutus PlanMill-toiminnanohjausjärjestelmään 36 s. + 3 liitettä 6.6.2012
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Yliopettaja Erja Nikunen Senior consultant/Manager Marjukka Niinioja

Työn tarkoituksena oli muuttaa PlanMill-toiminnanohjausjärjestelmä tukemaan paremmin monikansallisia yrityksiä ja heidän asiakkaitaan. Järjestelmä muutettiin tukemaan useaa oletusvaluuttaa, eli kirjanpitovaluuttaa, sekä käyttämään valuuttapareja valuuttakäsittelyssä. Myös järjestelmän lokalisointi muutettiin tukemaan aluekohtaisia kielitermejä ja formatointia. Tämä mahdollisti lakisääteisesti validien termien käytön ja helpotti valuuttamuunnoksia, kun laskutetaan ulkomaisia asiakkaita. Samalla oli hyvä mahdollisuus uudistaa laskuissa käytettävät laskupohjat dynaamisiksi ja helposti ylläpidettäviksi, koska laskupohjat ovat riippuvaisia kielestä, alueesta ja valuutasta.

Näillä kaikilla muutoksilla tavoiteltiin parempaa kilpailukykyä PlanMill Oy:lle globaaleilla toiminnanohjausjärjestelmämarkkinoilla, joissa myös isot kilpailevat yritykset ovat. PlanMill Oy:n paineet laajentua Euroopan ulkopuolelle kasvavat koko ajan, ja yrityksen laajenemismahdollisuudet ovat hyvät.

Projektin alun jälkeen huomattiin tarve kehittää yrityksen sisäistä kehitysprosessia ja siitä johtuen koko projekti ei pysynyt aikataulussaan. Tästä johtuen projekti pilkottiin pienempiin osiin, joista osa vietiin jo tuotantoon, mutta loput viedään tuotantoon vasta myöhemmin. Tuotantoon vietyjä osia olivat laskupohjamuutokset ja kielitermien lokalisoinnin parantaminen.

Avainsanat	PlanMill, toiminnanohjausjärjestelmä, PSA, valuutta, kirjanpito, laskutus, lokalisointi

Metropolia

Abstract

Author(s) Title Number of Pages Date	Olli Niinioja Implement support for localization and currency pairs into PlanMill PSA –system. 36 + 3 6 Jun 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Erja Nikunen, Principal Lecturer Marjukka Niinioja, Senior consultant/Manager

The aim of this study was to implement support for localization and currency pairs into PlanMill PSA-system. Before the system supported only a single base currency in an PlanMill instance. In a global multi-company world this is a very narrow approach and reduces PlanMill's ability to grow world wide. This also requires including support for region and language specific terminology.

Changing all currency handling to use currency pairs, instead of triangulating all exchange through base currency, helps PlanMill to allow easy exchange rating between most used currencies in the world. This affects directly to how PlanMill handles invoice templates and enables templates to be changed to be more dynamic and maintainable.

With these changes PlanMill hopes to gain better competitivity in global PSA markets where other big players are. The pressure to grow outside Europe is increasing and PlanMill has a great potential for it.

During the project we noticed a need to develop our internal development process which directly affected the project schedule. Due the lengthening of the project it was decided that the project will be split into smaller parts and put to production separately. Parts that new have been put to production include dynamic invoice template changes and support for regional language terms.

Keywords	PlanMill, PSA, currency, accounting, invoicing, localization

Sisällys

Lyhenteet

BDD	Behavior Driven Development. Ketterä testaus- ja kehitysmenetelmä.
CRM	Customer Relationship Management. Asiakkuudenhallinta.
ERP	Enterprise Resource Planning. Perinteinen yritysten toiminnan-ohjausjärjestelmä.
HRM	Human Resource Management. Henkilöstöhallinta.
IETF	Internet Engineering Task Force. Internetin laatua ylläpitävä yhteisö.
JAR	Java ARchive. Javan arkistotiedostoformaatti, eli kirjasto.
jBehave	BDD-testiajuri Javalle.
JSP	JavaServer Pages. Javaan perustuva menetelmä WWW-sivujen luontiin.
l10n	Localization. Lyhenne englanninkielisestä termistä, l – 10 merkkiä – n.
pip	percentage in point. Finanssialalla käytetty käsite vaihtotarkkuudesta.
PM	Project Management. Projektinhallinta.
POM	Project Object Model. Mavenin XML-dokumentti, joka kuvaa kehitettävää Java-projektia.
PSA	Professional Service Automation. Toiminnanohjausjärjestelmän variaatio palveluyrityksille.
SaaS	Software As A Service. Ohjelmisto palveluna.
SSL	Secure Socket Layer. Salausprotokolla.
TDD	Test Driven Development. Ketterä kehitysmenetelmä testauksella.

XML	Extensible Markup Language. Standardoitu merkintäkieli.
XSL	EXtensible Stylesheet Language. Tyylimäärittely XML-dokumentille.
XSL-FO	Extensible Stylesheet Language Formatting Objects. XML-dokumentin sivumuotoinen esitystapa yhdessä sisällön kanssa.

1 Johdanto

Tämän projektin tilaajana oli PlanMill Oy. Yritys on myös nykyinen työnantajani.

Pienessä mittakaavassa ja yhdessä maanosassa yhden oletusvaluutan (kirjanpitovaluutan) käyttö yrityksen rahavirran käsittelyssä on riittävää. Mutta kun yritys laajenee esimerkiksi tytäryrityksenä toiseen maahan, tarvitsevat kunkin maan maayhtiöt paikallisen kirjanpitovaluutan lainsäädännöstä ja käytännön tarpeesta johtuen. Usean valuutan käyttö konsernin sisällä sen eri yrityksissä vaikeuttaa koko konsernin rahavirtojen käsittelyä ja etenkin raportointia. Usein konsernin sisällä käytetään yhtä yhteistä raportointivaluuttaa. Kukin konsernin yhtiö käyttää aina omaa kirjanpitovaluuttaansa tietojen siirtämiseen yhtiökohtaiseen kirjanpitoon ja saattaa laskuttaa asiakkaitaan kirjanpitovaluuttansa lisäksi myös muissa valuutoissa. Laskutettava yritys voi sijaita myös toisessa maassa kuin laskuttava yritys tai voi haluta muusta syystä laskunsa muussa valuutassa. Tällöin konsernin toiminnanohjausjärjestelmän tulisi pystyä tukemaan useita kirjanpitovaluuttoja ja valuuttapareja rahavirtojen käsittelyssä, jotta turhilta valuuttamuunnoksilta vältyttäisiin. Sen pitää myös selviytyä pakollisesta kulttuuri- ja maakohtaisesta terminologiasta laskutuksessa.

Tällä projektilla oli tarkoitus parantaa PlanMill Oy:n kilpailukykyä Euroopan ulkopuolella lisäämällä PlanMill-sovellukseen tuki lokalisoinnille, valuuttapareille ja usealle oletusvaluutalle. Tarkoituksena oli lisätä helppokäyttöinen asiakkaan itsensäkin ylläpidettävä lokalisointimahdollisuus tukemaan eri alueiden markkinoiden terminologiaa. Toinen tärkeä asia oli pitää asiakasyrityksen rahamäärät sen omassa kirjanpitovaluutassaan ja tehdä kaikki tarvittavat valuuttamuutokset valuuttaparien kautta. Muutos antoi tuen myös usealle omalle yritykselle koko konsernin yhteisessä instanssissa, eli myös tietokannassa tulee esiintymään useassa kirjanpitovaluutassa olevia rahamääriä. Tämä vaikutti suoraan raportointiin ja laskutukseen, joita kumpaakin jouduttiin korjaamaan, jotta ne toimivat muutetussa järjestelmässä.

Laskutuksessa tärkeää on pitää laskutettu määrä kiinteänä laskun valuutassa, eikä se saisi muuttua minkään valuuttakurssivaihtelun seurauksena. Siksi projektissa tehtiin myös parannus hyväksyttyjen myyntitilausten ja laskujen käsittelyssä tallentamalla valuutta, päivämäärä ja valuuttamäärä erilliseen tietueeseen helppoa uudelleenhakemista

varten. Koska laskujen terminologia on myös tärkeää, ja suurelta osin lakisääteistä, niin siksi uusille kieli-maa-yhdistelmille on aikaisemmin tarvittu oma laskupohja. Tämän projektin lopputuloksena turhat toisiaan kopioivat laskupohjat on muutettu pääsääntöisesti parametroitaviksi ja helposti ylläpidettäviksi dynaamisiksi pohjiksi, jotka voi vaivatta räätälöidä myös asiakkaan tarpeisiin. Samat muutokset tehtiin myös muihin PDF-raporttipohjiin ja Finvoice-pohjaan, joka on Suomessa käytetty sähköinen verkkolaskustandardi.

2 PlanMill

PlanMill Oy on pk-yritys, joka on vuodesta 2001 toimittanut internetpohjaista SaaS (Software as a Service) -toiminnanohjaussovellusta. SaaS tarkoittaa pilvipalvelua, jossa sovellusta käytetään verkon välityksellä sopivalla asiakasohjelmistolla, joka tässä järjestelmässä on normaali WWW-selain. PlanMill-sovelluksen juuret ovat Nokian sisäisessä projektinhallintatyökalussa, joka on omistussuhteiden muutosten myötä yhtiöitetty omaksi yhtiökseen vuonna 2001 ja siirtynyt nykyisen johdon omistukseen vuonna 2006. PlanMill-sovelluksella on yli 10 000 päivittäistä käyttäjää yli 25 maasta. Sovellus sisältää monia ERP-toimintoja kuten CRM:n (asiakkuudenhallinta), HRM:n (henkilöstöhallinta), PM:n (projektinhallinta), tuotehallinnan, tuntikirjausten seurannan ja laskutuksen. Koska sovellus toimii web-selaimella pilvessä, ei asiakaskohtaista tuotantoympäristöä tarvita. Suurin osa sovelluksen lisensseistä myydäänkin juuri pilvipalveluun. PlanMill Oy tarjoaa myös on-premise-ratkaisua, jolloin asiakas itse ylläpitää omaa tuotantoympäristöään.

Keskeisimpiä PlanMill-sovelluksen vahvuuksia ovat automaattinen laskujen generointi projektin tuotosta ja myyntitilauksista näin tukien koko myyntiprosessia. Laskutuksen ajankohdallinen ennustaminen tapahtuu laskutuserien automaattisen generoinnin avulla. Automaattinen tuntiraporttien hinnoittelu ja tuntiraporttien, kulujen ja ostojen kerääminen automaattisesti laskutuserille mahdollistaa myös projektien tuottavuuden reaaliaikaisen ennustamisen.

PlanMill Oy:n asiakkaat ovat pääsääntöisesti asiantuntijapalveluita tarjoavia pk-yrityksiä. Tähän segmenttiin kuuluvat muun muassa IT-palveluita, IT-sovelluskehitysprojek-

teja, mainos- ja media-alan palveluita, taloushallinnon palveluita sekä arkkitehti- ja suunnittelualan palveluita tarjoavat yritykset.

2.1 Toiminnanohjausjärjestelmä

Yrityksen toiminnanohjausjärjestelmä on sovellus, jolla hallitaan ja monitoroidaan yrityksen toimintaa. Toiminnanohjausjärjestelmät yleensä perustuvat modulaarisuuteen, eli helppoon lisäominaisuuksien käyttöönottoon. Toiminnanohjausjärjestelmän yleisiä toimintoja ovat

- CRM eli asiakkuudenhallinta
- HRM eli henkilöstöhallinta
- PM eli projektinhallinta
- kirjanpito
- reskontra
- varastonhallinta
- tuntikirjausten seuranta
- tuotannonohjaus
- laskutus
- raportointi.

Perinteistä toiminnanohjausjärjestelmää kutsutaan lyhenteellä ERP eli Enterprise Resource Planning. Se on yleensä laaja kokonaisuus erilaisia integroituja toimintoja, kuten edellinen lista lajittelee. (Toiminnanohjausjärjestelmä, 2011.)

ERP-ohjelmistot voivat olla joko lokaalissa ympäristössä ajettavia ja asiakkaan itse ylläpitämiä järjestelmiä tai pääteohjelmistolla käytettäviä etäpalveluita, jolloin palvelun fyysisestä ylläpidosta vastaa palveluntarjoaja. Jälkimmäinen vaihtoehto tunnetaan yleisesti nimellä pilvipalvelu. Perinteiset ERP-kokonaisuudet keskittyvät enemmän tuotantoautomaatioon ja kirjanpitojärjestelmiin, missä raakaa tietoa muotoillaan järjestelyrakenteisiin, jotka noudattavat koko yrityksen kirjanpidollista näkymää. Nämä rakenteet ovat käteviä tehtäessä pääkirjaa tai muuta korkeamman tason kirjanpitoa. (Melik, 2002, s. xiii.)

Tunnettuja ERP-järjestelmien tuottajia maailmalla ovat muun muassa SAP, joka tekee pääsääntöisesti asiakkaan itse ylläpitämiä (on-premise) ratkaisuja isoille teollisuusyrityksille. PlanMill Oy:llä on asiakkaita, jotka käyttävät SAP:ia ERP-toimintaan, mutta käyttävät rinnalla myös PlanMilliä projektinhallintaan ja PSA-järjstelmänä (Professional Services Automation). Varsinaisia PlanMillin kilpailijoita ovat esimerkiksi Severa, Netvisor, Netsuite, Freshbooks ja OpenAir, jotka kaikki keskittyvät juuri PSA-toimintoihin. Tähän joukkoon voisi myös lisätä Salesforcen, jonka tunnetuin tuote on CRM-pilvipalvelu.

2.2 Professional Services Automation

Toiminnanohjausjärjestelmistä on myös useita variaatioita, kuten PSA eli Professional Service Automation, joka on kevyempi toiminnanohjausratkaisu asiantuntijayrityksille. Se keskittyy pääsääntöisesti tilaus- tai tuntipohjaisten laskutettavien projektien ohjaamiseen, seurantaan ja laskutukseen (PSA-järjestelmä, 2011). PlanMill-sovellus on lähempänä PSA-järjestelmää kuin ERP:tä, vaikkakin sovelluksen ominaisuuksiin kuuluu esimerkiksi tuotehallinta, mikä PSA-järjestelmistä yleensä puuttuu.

Toiminnanohjausjärjestelmät ovat olemassa yrityksen toiminnan optimointia varten. Tällä parannetaan yrityksen tuottavuutta ja parannetaan kilpailukykyä nykyään niin kilpailutetuilla globaaleilla markkinoilla. PSA-järjestelmän tarjoamat ratkaisut automatisoivat ja integroivat projekti- ja palvelualanyritysten ydinliiketoiminnan prosesseja tarkoituksenaan kasvattaa tehokuutta projektien suunnittelussa, budjetoinnissa, luomisessa, henkilöittämisessä, aikataulutuksessa, implementoinnissa ja laskutuksessa yritysmaailmassa. (Melik, 2002, s. xiv.)

Ydinliiketoiminnan helpottuminen tarkoittaa käytännössä suoraan yrityksen tuottavuuden parantumista. Yrityksen tuottavuus todetaan kirjanpidossa, jonka kirjanpitovaluuttana on yrityksen maan valuutta. Tuottavuutta voidaan myös monitoroida järjestelmän erilaisilla raporteilla, jotka pääsääntöisesti käsittelevät jotain valuutassa mittavaa.

2.3 PlanMill-järjestelmä

PlanMillin palvelinsovellus, tuttavallisemmin backend, on koodattu Javalla, ja sen käyttö tapahtuu Tomcat-sovelluspalvelimella. Sovellus käyttää JSP-tekniikkaa (JavaServer pages) tiedon välittämiseen asiakkaan selaimelta palvelimelle. Palvelimelta asiakkaalle tieto tuodaan XSL-sivutyylipohjilla luoduilla dynaamisella XHTML-sivuilla.

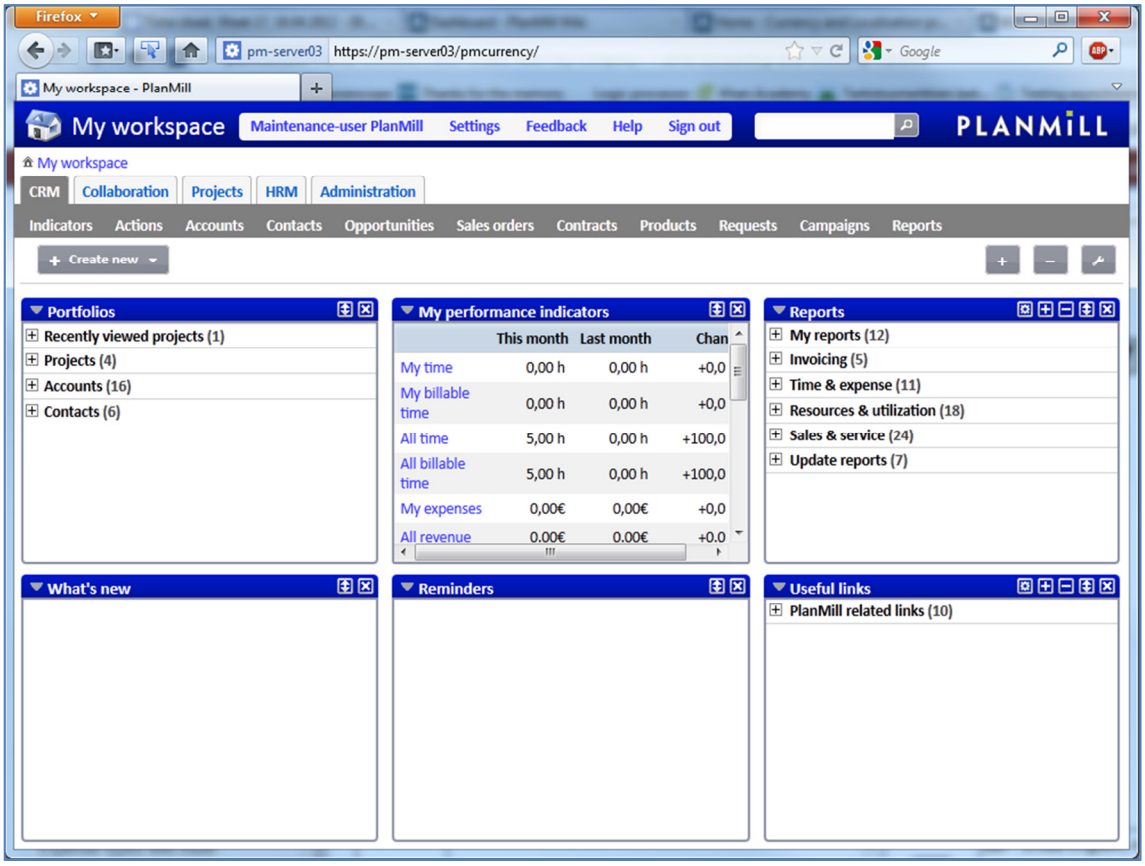
Sovellus on klusteroidussa ympäristössä, ja tällä hetkellä sovelluspalvelimia on kaksi, joskin niitä voi olla käytännössä rajaton määrä. Sisään tulevaa suojattua SSL-liikennettä (Secure Socket Layer) eri sovelluspalvelimille jakaa Ciscon-kuormanjakaja (load balancer). Jokaisella palvelimella on myös oma liikennettä ohjaava Nginx-niminen välityspalvelinsovellus (reverse proxy), joka ohjaa HTTP-kyselyt Tomcatille sovelluksen käsittelylogiikkaan ja palauttaa saadun tuloksen. Nginx mahdollistaa myös helpon IP-suodatuksen, eli esimerkiksi vain tietyn IP-osoitteen, tai osoiteavaruuden, pääsyn tiettyyn PlanMill-instanssiin.

Cloud-ympäristössä, eli PlanMillin pilvipalvelussa, PlanMill-instanssi kerrotaan selaimelle URL-osoitteessa (Uniform resource locator), esimerkiksi https://production.domain.com/**planmill_instance**. Tällä hetkellä PlanMill-instansseja on tuotantoympäristössä yli sata kappaletta. Cloud-ympäristössä asiakkailla on aina käytössään uusin versio, johon päivityksiä voidaan tehdä periaatteessa lähes jatkuvasti. Käytännössä päivityssykli osuu kuitenkin pienimpien haittavaikutusten vuoksi useimmiten kuukauden 20.-25. päivien väliin, koska suurin osa asiakkaista laskuttaa aktiivisimmin kuun 1.-20. päivien välisenä aikana ja suurin osa tunti- ja kuluraportoinnista taas tapahtuu kuukauden 25.-30. päivien välisenä aikana.

Osalla asiakkaista järjestelmä on asennettuna heidän omille palvelimilleen (on-premise), jolloin heillä on käytössä yleensä vanhempia versioita järjestelmästä. Näille asiakkaille järjestelmän uudet versiot ja korjaukset päivitetään vain erikseen sovitta-

essa. On-premise-asennusten päivitys on yleensä aina iso työ johtuen vanhan version räätälöinneistä, joissa on joskus jopa lähdekoodiinkin tehtyjä muutoksia.

Alla löytyy esimerkkikuva projektin kehitysinstanssin etusivusta. Selaimena toimii Mozilla Firefoxin versio 12. Järjestelmä ulkoasu pyritään pitämään mahdollisimman samanlaisena kaikilla eri seilain ja käyttöjärjestelmä -kombinaatioilla. Käytännössä tämä on kuitenkin melkein mahdotonta, koska eri selainvalmistajat eivät noudata standardeja täysin.



Kuva 1. PlanMill-sovelluksen etusivu testiympäristössä. Kuvan ikkunaa on supistettu.

Kuvasta voi huomata, että terminologiaa on paljon ja jo etusivulla näytetään tärkeitä mittareita raportoiduista tunneista, kuluista ja rahavirroista. Nämä kaikki kuuluvat käyttäjäyrityksen lokalisaation piiriin ja niiden formatoinnin tulee olla helposti vaihdettavissa toiseksi. Yhdessä instanssissa konsernin eri yritykset, tai jopa yksittäiset käyttäjät, voivat valita haluamansa lokalisaatioasetukset helposti asetuksista.

3 Valuutat ja lokalisointi

Valuutalla tarkoitetaan valtion tai alueen rahaa ja rahayksikköä, jota käytetään vaihtokauppavälineenä. Valuuttojen välillä käytetään vaihtokurssia, joka kertoo valuutan arvon suhteessa toiseen valuuttaan. Kahden eri valuutan välisestä vertailusta käytetään termiä *valuuttapari*. Valuuttaparien vaihtokurssit muuttuvat yleensä päivittäin. Nykyisten kehittyneiden maiden valuutat ovat niin sanotusti kelluvia, eli niiden arvo ei ole kiinteä, vaan sen vaihtokurssi suhteessa muihin valuuttoihin määräytyy valuuttakaupalla. Valuuttakauppa tapahtuu asiakkaan ja pankin välillä tai kahden pankin välillä. (Valuutta, 2012.)

Vaihtokurssit vaikuttavat myös toiminnanohjausjärjestelmän valuuttakäsittelyyn. Joitain järjestelmän tietoja, kuten laskuja muussa kuin kirjanpitovaluutassa, joudutaan sitomaan tietyn päivän vaihtokurssiin. On myös tärkeää pystyä muuntamaan yhden järjestelmäinstanssin eri kirjanpitovaluutat keskenään, ja vielä erikseen raportointivaluutalle.

Valuuttojen nimien lyhenteet ovat määritelty ISO 4217 -standardissa (ISO 4217, 2012) ja tulevat yleensä suoraan valuutan maasta ja nimestä, kuten USD (United States Dollar), GBP (Great Britain Pounds, pound sterling) ja JPY (Japanese yen). Tosin poikkeuksiakin on kuten Euroopan euro (EUR). PlanMill-järjestelmä pystyy jatkossa käsittelemään kaikkia valuuttoja ja valuuttapareja, jotka on määritelty ISO-standardin mukaisesti.

3.1 Valuuttaparit

Valuuttaparit (currency pairs) ovat de facto -käsite kaikessa valuuttakäsittelyssä. Kaikki valuuttakauppa tapahtuu aina valuuttapareissa. Valuuttaparissa valuutan hinta on määritetty suhteessa toiseen valuuttaan. Valuuttapari kertoo, kuinka paljon jälkimmäistä valuuttaa tarvitaan ostaessasi yhden yksikön ensimmäistä valuuttaa. Myös toiminnanohjausjärjestelmän valuuttojen muunnos on paras käsitellä valuuttapareina. Tunnettuja suuria valuuttapareja ovat

- EUR/USD (Fiber)
- EUR/GBP (Chunnel)
- GBP/USD (Cable)
- USD/CAD (Loonie)
- AUD/USD (Aussie)
- GBP/JPY (Geppie)
- NZD/USD (Kiwi).

Valuuttaparien perässä sulkeissa olevat lisänimet ovat maailmanlaajuisesti tunnettuja lempinimiä paljon vaihdetuille valuutoille. Esimerkiksi *Cable* on saanut nimensä ajalta, kun Atlantin valtameren pohjalla kulkeva kaapeli synkronoi GBP/USD:n valuuttakurssia Lontoon ja New Yorkin välillä. Valuuttaparien lempinimiä käytetään tosin yleensä vain valuuttakaupassa. (Forex Basics, 2012; Currency pair, 2012.)

Valuuttakurssin pienintä vaihtoyksikköä kutsutaan nimellä *pip* (percentage in point). Valuuttakaupassa se on pienin mahdollinen vaihtoyksikkö. Esimerkiksi EUR/USD:n pip on 0,0001 kun taas USD/JPY:n pip on 0,01. Tämä johtuu Yhdysvaltain dollariin ja Japanin jenin suuresta vaihtokurssista, joka on tänään, 18.4.2012, noin 80,59 jeniä dollarilta, eli USD/JPY:n vaihtokurssi on 80,59 ja vastaavasti EUR/USD vaihtokurssi on 1,3128 (Currency Converter, 2012). Käänteisen valuuttaparin JPY/USD:n pip on 0,0001 ja vaihtokurssiksi laskemalla 1/80,59 saadaan 0,0124. Tähän suuntaan tarkkuuden on oltava suurempi, koska muuten muunnoksessa katoaisi liikaa tietoa. Esimerkiksi jos sama lasku tehtäisiin tarkkuudella 0,01, niin tulokseksi saataisiin vain 0,01, jonka uudelleenkääntäminen antaisikin alkuperäiseksi vaihtokurssiksi 100,00.

3.2 Vaihtokurssit

Maailmanmarkkinoilla valuuttakauppa käydään pankkien välillä, joko suoraan, tai erilaisten elektronisten kaupanvälitysalustojen kautta. Valuuttamarkkinat eivät ole sää-

deltyjä eivätkä keskitettyjä, joten vaihtokurssit voivat olla erisuuruiset eri pankkien välillä, eikä vaihdettaessa ole takuuta, että saman summan saa takaisin takaisinvaihdossa (Interbank foreign exchange market, 2012). Tästä johtuen myös valuuttamuunnokset valuutasta toiseen tulee tehdä aina yhden valuuttaparin kautta. Jos valuuttamuunnoksessa käytetään välissä kolmatta valuuttaa, eli triangulaatiota (triangulation), on lopputuloksena hyvinkin todennäköisesti eri loppusumma kuin käytettäessä vain itse valuuttaparin vaihtokurssia (Concept of Currency Cross Triangulation , 2010). Esimerkiksi voidaan ottaa USD:n vaihto euroiksi kahdella eri tavalla. Ensimmäisessä vaihdetaan välillä USD/GBP ja sitten GBP/EUR ja toisessa vaihdetaan suoraan USD/EUR. Kummallakin valuuttaparilla vaihtoyksikkö (pip) on 0,0001.

```
USD/GBP kurssi on 0,6197
GBP/EUR kurssi on 1,2219
USD/EUR kurssi on 0,7573

Esimerkki 1:
100 USD * 0,6197 = 61,97 GBP
61,97 GBP * 1,2219 = 75,7211 EUR

Esimerkki 2:
100 USD * 0,7573 = 75,73 EUR
```

Esimerkkikoodi 1. Triangulaatio, eli valuuttamuunnos kolmannen valuutan kautta. Kurssit on otettu 26.4.2012 lähteestä OANDA:n reaaliaikaiset kurssit (Live Exchange Rates, 2012).

Yllä olevasta esimerkistä huomataan, että käyttämällä kolmatta valuuttaa välissä kadotetaan 100 USD valuuttamäärällä melkein yksi sentti (0,01 EUR) muunnoksessa. Laskennassa on käytetty oikeita valuuttakursseja ja tarkkuuksia. Tästä voi päätellä, että esimerkin kursseilla yhden miljoonan (1M USD) vaihdossa voidaan kadottaa jo 100 euroa.

Myös EKP, eli Euroopan keskuspankki, tarjoaa päivittäin yleisimpien valuuttojen vaihtokurssit suhteessa euroon. WWW-käyttöliittymän lisäksi tämä palvelu tarjotaan myös XML-tietueena helppoa sovellusintegraatiota varten. Valitettavasti EKP ei tarjoa palvelussaan Euroopan ulkopuolisia valuuttapareja. Esimerkki tällaisesta palvelusta, joka tarjoaa kaikki tärkeimmät valuuttaparit, on OANDA:n Live Exchange Rates (Live Exchange Rates, 2012). Tämä palvelu toimii ilmaiseksi selaimella yksityiskäyttöön, mutta yrityksille suunnattu vaihtokurssipalvelu helposti integroitavassa muodossa on maksullinen.

```
<?xml version="1.0" encoding="UTF-8"?>
<gesmes:Envelope
   xmlns:gesmes=http://www.gesmes.org/xml/2002-08-01
   xmlns="http://www.ecb.int/vocabulary/2002-08-01/eurofxref">
    <gesmes:subject>Reference rates</gesmes:subject>
    <gesmes:Sender>
        <gesmes:name>European Central Bank</gesmes:name>
    </gesmes:Sender>
    <Cube>
        <Cube time="2012-04-20">
            <Cube currency="USD" rate="1.3192"/>
            <Cube currency="JPY" rate="107.81"/>
            <Cube currency="GBP" rate="0.81875"/>
        </Cube>
    </Cube>
</gesmes:Envelope>
```

Esimerkkikoodi 2. Osa EKP:n XML-dokumentista sisältäen päivän valuuttakursseja suhteessa euroon. (ECB Euro FX, 2012).

Käytännössä euron eri valuuttapareista voidaan myös luoda kaikki mahdolliset valuuttaparikombinaatiot, kunhan huomioidaan virhemarginaali, joka vaihtelee välillä paljonkin riippuen päivän kaupasta. PlanMillissä päivän kurssit tulevat voimaan vasta seuraavan päivän puolella johtuen EKP:n kurssien julkaisuajankohdasta, joka osuu iltapäivälle. Siksi kurssit eivät sovellu käytettäväksi saman päivän laskutoimituksissa, jotka on voitu tehdä esimerkiksi aamupäivällä jo ennen kurssien päivittymistä.

3.3 Lokalisointi

Lokalisoinnilla tarkoitetaan tuotteen, sovelluksen tai dokumentin sisällön muutosta vastaamaan kieli-, kulttuuri- tai muita vaatimuksia valituilla markkinoilla. Englanninkielessä lokalisoinnista voidaan käyttää termiä l10n, jossa 10 merkitsee l:n ja n:n välissä olevaa merkkimäärää. Yleensä lokalisointi käsitetään synonyyminä käyttöliittymän ja dokumenttien kielikäännöksille. Se on kuitenkin monimutkaisempi käsite, jolla voidaan esimerkiksi tarkoittaa seuraavien asioiden muuntamista:

- numero-, päivä- ja aikaformaatti
- valuuttakäyttö
- näppäimistön käyttö
- tietojen kerääminen ja järjestäminen

- symbolit, ikonit ja värit

- tekstin ja grafiikan yhdistäminen asioihin, tekoihin tai ideoihin, jotka kulttuurikohtaisesti voidaan tulkita väärin

- lakivaatimukset ja normit.

Lokalisointi voi myös vaatia kokonaisvaltaista logiikan, ulkoasun tai esittämisen uudelleenajattelua, kun liiketoiminta valituilla markkinoilla eroaa omasta kulttuurista. (Localization vs. Internationalization, 2005.)

Viime aikoina PlanMillillä on ollut enemmän paineita laajentua myös Euroopan ulkopuolelle. Jo nyt PlanMillin asiakkailla on asiakkaita ympäri maailmaa, joita he myös laskuttavat PlanMill-sovelluksella. Aikaisemminkin on ollut tieto, missä maassa mikäkin asiakasyritys sijaitsee ja mikä kieli maassa on, mutta kielitermit on tallennettu vain kielelle ilman maakohtaista terminologiaa. Tämä on ollut puutteellinen lähestymistapa globaalissa markkinataloudessa, koska usein kohdemarkkinoiden terminologia eroaa yrityksen omasta kulttuurista, vaikka kieli olisikin sama. Asia ei ole yksinkertainen edes Suomen osalta. Esimerkiksi jos suomalainen yritys lähettää laskun ruotsalaiselle asiakkaalle, se joutuu tekemään laskun suomenruotsiksi, jotta yritystunnuksen käännökseksi tulisi oikein *FO-nummer* (*företags- och organisationsnummer*), kun taas Ruotsissa validi termi on pelkästään *organisationsnummer*.

Englanninkielinen termi *Locale* tarkoittaa tietotekniikassa kokoelmaa käyttäjän kielestä, maasta ja mahdollisesta variantista, jonka käyttäjä haluaa nähdä käyttöliittymässään. Yleensä *locale*-tunniste sisältää ainakin kielitunnisteen ja aluetunnisteen. POSIX-alustoilla, kuten Unixissa, Linuxissa ja muissa vastaavissa, *locale*-tunnisteet on määritelty kuten IETF:n (Internet Engineering Task Force) BCP 47-määritellyissä kielikoodeissa (BCP 47, 2009), mutta variantti muuttuja on määritelty hieman eri tavalla ja merkistötieto on lisätty tunnisteeseen. Formaatti on [kieli[_alue][.merkistö][@muuttuja], joten esimerkiksi Australian englannille se olisi en_AU.UTF-8. (Locale, 2004.)

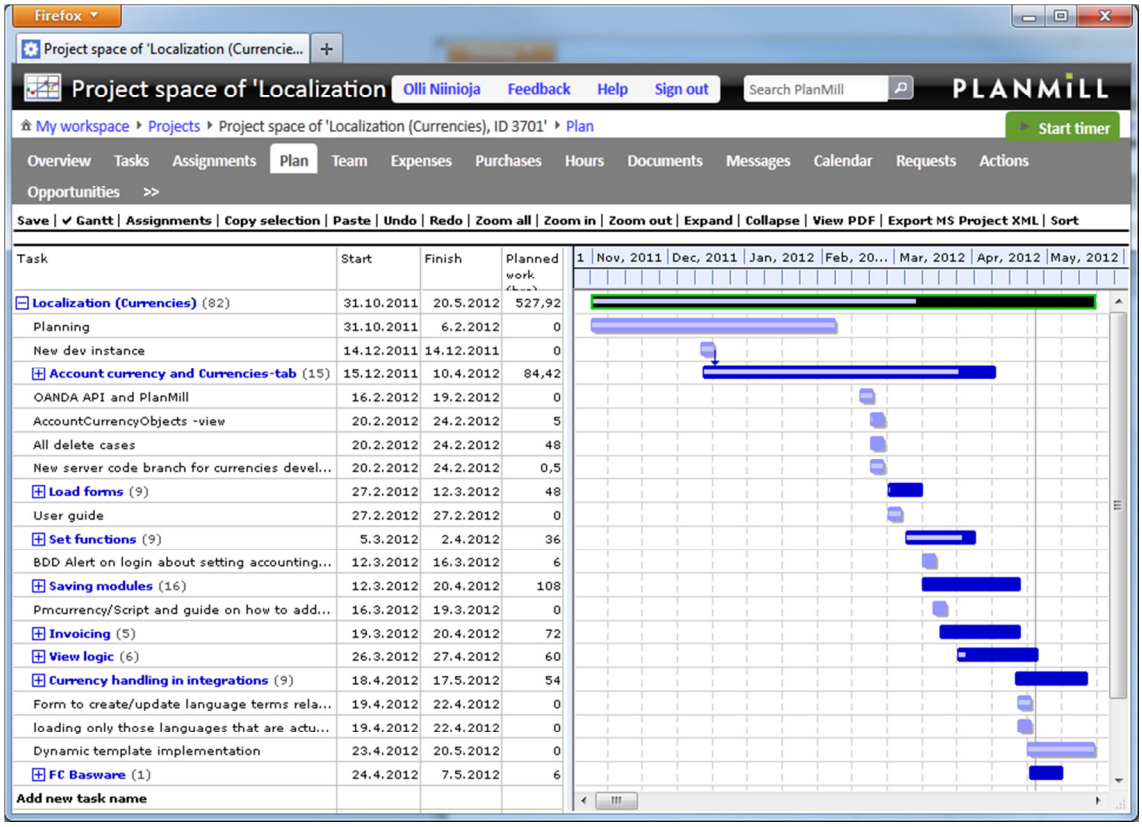
Lokalisointia käytetään paljon WWW-sivuilla, koska sivujen selaajat voivat sijaita millä mantereella ja missä maassa tahansa. Tämä mahdollistaa helpon tarttumapinnan ja miellyttävän käyttökokemuksen käyttäjälle ja näin parantaa mahdollisuuksia saada

käyttäjä käymään sivuilla uudestaankin. Perinteiselle (X)HTML-sivun lokalisoinnille ei ole yhtä oikeaa tapaa, mutta W3C on tehnyt listan hyvistä tavoista, joita kannattaa noudattaa (Internationalization Best Practices: Specifying Language in XHTML & HTML Content, 2007). Vaikka PlanMill-sovellus toimiikin selaimella ja näkyvät sivut ovat XSL-tyylipohjilla luotuja XHTML-sivuja, oli projektissa kuitenkin tarkoitus parantaa lokalisointia palvelinpuolella.

4 Projektin toteutus

4.1 Projektin suunnittelu

Alustavasti projektille ei ollut mitään selvää tiettyä päämäärää kuin mahdollistaa sovelluksen helpompi käyttöönotto ja käyttö myös Euroopan ulkopuolella, ja sitä lähdettiin kartoittamaan monin tavoin. Tärkein päämäärä oli luoda lisäarvoa asiakkaille ja siksi asiakkaiden käyttökokemus on tärkeä. Vuosien varrella järjestelmää on kehitetty paljolti asiakkailta saadun palautteen perusteella. Projektista luotiin seuraavanlainen projektisuunnitelma, joka on elänyt projektin myötä.



Kuva 2. Projektisuunnitelman päätehtävien luonnos gantt-kaaviona. Projektisuunnittelussa on käytetty PlanMill-sovellusta.

Projektissa pyrittiin käyttämään mahdollisimman ketterää kehitysmenetelmää, mutta koska valuutta- ja terminologia asia oli koko järjestelmän kattava, tiedettiin jo alusta alkaen, että minkä tahansa muutoksen vaikutukset koko järjestelmään tällä alueella ovat suuret ja vaikuttavat suureen osaan asiakkaista. Niinpä projektin alussa tehtiin suuri määrä tutkimustyötä muiden järjestelmien tavasta toimia ja perehdyttiin aihealuetta koskeviin tutkimukseen. Myös PlanMill-järjestelmän olemassa olevaa koodia ja toiminnallisuutta, unohtamatta asiakkaille tehtyjä räätälöintejä, oli tutkittava koko projektin ajan. Suurin vaikeus projektin aikana oli löytää ne kohdat järjestelmästä, joita muokkaamalla voitaisiin tehdä mahdollisimman pienellä riskillä mahdollisimman suuri lisäarvo asiakkaille.

Projektin kehityksen alkuvaiheilla pyrittiin käyttämään mahdollisimman ”oikeaoppista” Scrum- tai oikeammin Scrum-ban menetelmää stand up-palavereineen ja Kanban-tauluineen (Scrum, 2012), mutta sen käyttö väheni ajan myötä lähinnä resurssitilanteiden muutoksen ja projektiin sisältyneen tutkimuksen suuren osuuden vuoksi. Stand up-palavereita jatkettiin lähes projektin loppuun asti.

Käyttäjätarinat

Projektia lähdettiin kehittämään käyttäjätarinoilla (user stories), jotta tärkeimmät käyttötarkoitukset saatiin kartoitettua. Hyvä tapa simuloida palautteen kuvailemaa käyttökokemusta on kirjoittaa käyttäjätarinoita käyttäjän näkökulmasta. Käyttäjätarinat ovat helppo tapa aloittaa määrittely siltä pohjalta, mitä käyttäjä haluaa tehdä. Näillä tarinoilla saadaan rajattua yleisimmät järjestelmän toiminnan skenaariot, jotka voidaan jo kehittää. Ketterässä kehityksessä käyttäjätarinoita voidaan luoda sitä mukaa, mitä edellisiä on saatu toteutettua ja liitettyä järjestelmään. Kyseinen kehitys on tyyliltään iteroivaa ja näin tehdäänkin esimerkiksi Scrum–projektinhallinnassa. Käyttäjätarinoita on helppo hallita esimerkiksi taulukkona, jossa sarakkeet kertovat käyttäjän roolin, tarpeen, oletetun tuloksen ja kriteerin, jolla verifioidaan käyttötapauksen onnistuminen.

Taulukko 1. Esimerkkejä projektin backlogista poimituista käyttäjätarinoista.

As a...	I would like to...	so that I...	Acceptance criteria
Power User	Set the defult invoicing currency and other preferred currencies for a My company (Overrides Default)	so that (only?) these currencies will be available for users to select so that they will not accidentally select a currency that is not preferred by our financial department because of volatility or other high risks.	show currencies in select lists in preferred order 1. first the currency based on the country of my company/customer if included in the preferred currency list, 2. all other preferred currencies in alphabetical order, 3. all other currencies in alphabetical order
As a User	I want to add expense items in accounting currency of my company account	so that our company's payroll personnel can pay the expenses to me in my company's 'accounting currency'	If I have travel expenses or other items in different currency than my company accounting currency I want to give a rate with which the sum is converted to accounting currency. The rate doesn't need to be saved, it's one-time conversion only Only accounting currency entries are allowed

Projektissa haluttiin myös ensimmäistä kertaa aloittaa testaus ja dokumentointi jo projektin alkuvaiheessa. Käyttäjätarinat ovat hyvä dokumentointiväline, mutta ne kuitenkin ovat liian suurpiirteisiä tarkemmalle toiminnallisuudelle. Tähän ongelmaan totesimme hyväksi työkaluksi ”BDD:t” eli Behavior Driven Development -menetelmän mukaisesti testitapauksina kirjoitetut toiminnallisuuskohtaiset määrittelyt.

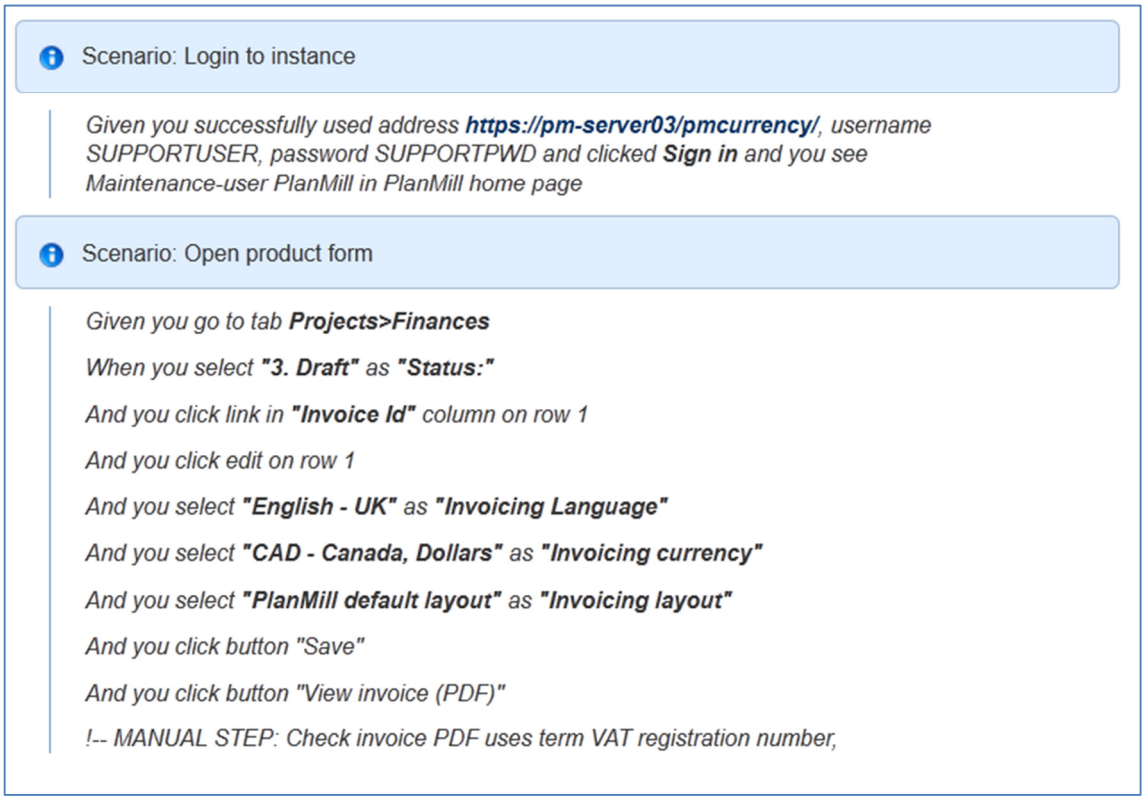
Behavior Driven Development

BDD, eli käyttäjätarinoihin perustuva ketterä kehitysmenetelmä, laajentaa tunnettua TDD-kehitysmenetelmää ottamalla huomioon käyttäjän näkökulman sovelluskehityksessä. TDD tarkoittaa kehitysmenetelmää, jossa yksikkötestit suunnitellaan ja toteutetaan ennen niillä testattavaa toimintoa. Yksikkötesti on yksinkertainen testi, jolla verifioidaan jokin toiminto.

BDD:t ovat looginen jatkumo käyttäjätarinoille. BDD:ssä käyttäjätarinat pilkotaan pienempiin tarinoihin, jotka dokumentoivat toiminnon ja toimivat samalla käyttöliittymätesteinä, eli toisin ajateltuna yksi tarina on myös toiminnon hyväksyntätesti (Acceptance testing, 2012). Tämä kaikki tehdään jokaiselle toiminnolle erikseen ennen itse toiminnon ohjelmoinnin aloittamista. Lisänä sovelluskehityksessä voidaan käyttää myös TDD-kehitysmenetelmää toiminnon koodin yksikkötestauksessa sovelluspuolella. Tätä kokonaisuutta voi ajatella sekoitettuna vesiputousmallina, jossa ensin aloitetaan määrittelystä ja suunnittelusta, mutta sitten tehdäänkin jo integraatiotestit, ja sitten ennen toiminnon ohjelmointia tehdään toiminnon tarkat yksikkötestit ja lopuksi ohjelmoidaan itse toiminnallisuus. Näin kehityksen aikana voidaan yksikkötesteillä ja integraatiotesteillä todentaa, milloin toiminto on valmis. (Behavior Driven Development, 2012.)

BDD-testaus käsitteenä ei ole sidottuna mihinkään ohjelmointikieleen, ja sillä onkin mahdollista testata lähes mitä tahansa sovellusta, kunhan BDD-testaustiedostoille löytyy sopiva ajuri, sovelluksella on avoin rajapinta tai testiohjelmalla on mahdollisuus kutsua sovelluksen käännettyä lähdekoodia. Hyvä esimerkki joustavasta BDD-testausajurista on *Cucumber*, joka toimii muun muassa Rubyn, Javan, .NET:in ja Flexin kanssa (Hellesøy, 2012). Tässä projektissa käytettiin BDD-ajurina Javassa toimivaa jBehave-pakettia, johon oli integroitu saman kehittäjän toteuttama Selenium WWW-ajuri.

BDD-testaus tapahtuu useilla, joskus jopa sadoilla, .story–tiedostoilla, jotka sisältävät selkokielellä käyttöliittymäjärjestelmän tai muun järjestelmän käyttäytymistoiminnallisuuden testauksen. Näitä tiedostoja ajetaan sopivalla ajurilla, joka tässä projektissa oli Javalla toimiva ja ohjelmoitava jBehave (jBehave, 2012). Jotta ajurilla saatiin testattua web-käyttöliittymää, tarvitsi se myös erillisen web-ajurin, esimerkiksi Seleniumin, joka emuloi tavallisen käyttäjän toimintoja omalla selaimellaan. Testaus toimii käytännössä tilakoneena, eli seuraava testi jatkaa siitä selaimen tilasta, mihin edellinen testi sen jätti. Tämä helpottaa huomattavasti monimutkaisten sovellusten testaamista kun jokaista testiä varten selainta ei tarvitse käynnistää uudestaan. Testit voivat olla myös riippuvaisia toisista testeistä, jotka ajuri osaa ajaa oikeassa järjestyksessä ennen ajettavaa testiä.

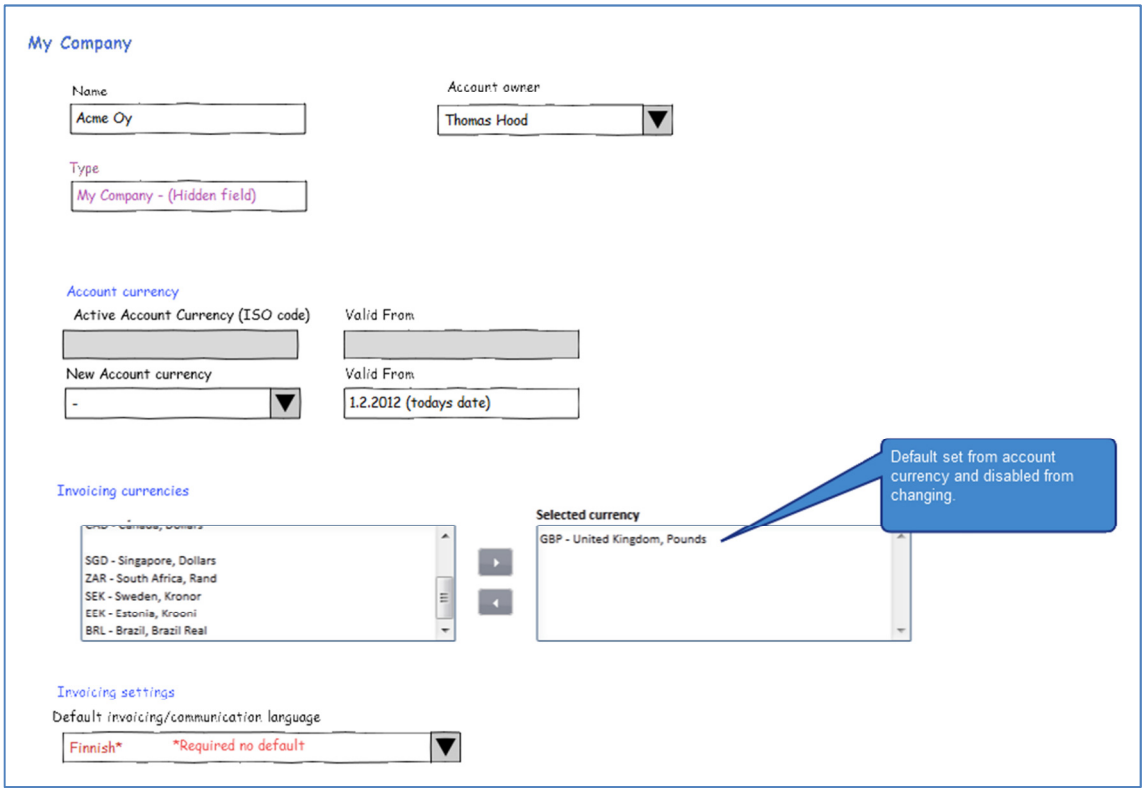


Kuva 3. Esimerkki PlanMillin wiki-sivulle talletetusta ajettavasta BDD:stä.

Lokalisointi käyttäjäyrityksille

Järjestelmän käyttäjien yrityksille (my companies) valitut asetukset vaikuttavat koko asiakasinstanssin toimintaan. Käyttäjäyrityksen maata käytetään nyt esimerkiksi suoraan laskun lokalisoinnin alueena, kun laskun kieli ja valuutta tulevat laskun laskupohjavalinnasta. Käyttäjäyrityksille haluttiin projektissa antaa myös tuki yrityksen

oman maan kirjanpitovaluutalle. Kirjanpitovaluutta on muutettavissa oman yrityksen lomakkeella. Lomakkeella voidaan myös valita yrityksen käyttämät laskutusvaluutat.



Kuva 4. Alustava kuva käyttäjäyrityksen lomakkeen laskutuspuolen käyttöliittymästä.

Lokalisointia tukevat formaattikäsittelyn muutokset päätettiin tehdä mahdollisimman syvälle Javan koodiin, jota jo kaikki moduulit järjestelmässä käyttävät. Tämä paikka oli FormatControl-luokan formatField-metodi, joka käytännössä käsittelee kaiken yrityslogiikkaluokkien kenttien formatoinnin. Tuki kielten maakohtaisille termeille variantteineen päätettiin vastaavasti upottaa syvälle FormatControl-luokan getLangText-metodiin, joka jo hakee Javan muistista oikean termin oikealle kielikoodille.

4.2 Kehitysympäristö

Yrityksessä on käytössä Java-kehittimenä Eclipse IDE. Versiohallintana sisäverkon palvelimella toimii Subversion. Kehityskoneilla versiohallinnan asiakassovelluksena on TortoiseSVN.

Projektissa päätettiin ottaa ensimmäistä kertaa kehityksessä käyttöön myös Maven, jolla helpotetaan huomattavasti Java-projektin ylläpitoa varsinkin kirjastoriippuvuuk-

sien, JAR:ien (Java ARchive), hallinnassa. Mavenin ydinideana on POM, eli Project Object Model, joka on XML-dokumentti ja sijaitsee lokaalissa tiedostojärjestelmässä projektin juuressa tiedostona nimellä pom.xml. POM määrittelee kaikki projektin riippuvuudet ja mahdolliset käytettävät liitännäiset (plugin) ja niiden asetukset. Maven tukee myös koko sovelluksen elämänkaarta, aina käännöksestä ja yksikkötestauksesta koko sovelluksen käyttöönottoon (deploy) ja integraatiotestaukseen. Supistettu versio PlanMillin POM:ista löytyy liitteenä 1. Tämä johtuu siitä, että PlanMill-sovelluksella on useita kymmeniä kirjastoriippuvuuksia kolmannen osapuolen kirjastoihin.

Maven ylläpitää omaa kirjastosäilytyspaikkaa (repository), jonka sisältämiin kirjastoihin kehittimessä luodaan linkitys (build-path). Yrityksen kehitysympäristön sisäverkossa on myös erillinen kirjastojen cache-palvelin, Nexus, joka sisältää jo entuudestaan haetut kirjastot ja mahdollistaa nopeamman kirjastohaun. Jos kirjastot eivät löydy lokaalista eikä sisäverkon säilytyspaikasta, haetaan ne Mavenin julkisesta säilytyspaikasta.

Mavenin käyttöönottoa sekä versionhallinnan trunk- ja branch-toimintojen laajempaa käyttöä oli suunniteltu jo jonkin aikaa, mutta vasta projektin aikana yritys siirtyi käyttämään useampaa rinnakkaista sovelluskehitystiimiä, jolloin rinnakkaisten projektien kehittäminen vaati käytännössä myös sovelluskehitysmenetelmien ja työkalujen kehittämistä sekä rinnakkaisten kehitysympäristöjen käyttöönottoa. Tälle projektille oli käytössä oma kehityshaara *pmcurrency* ja sovellusinstanssi /*pmcurrency*.

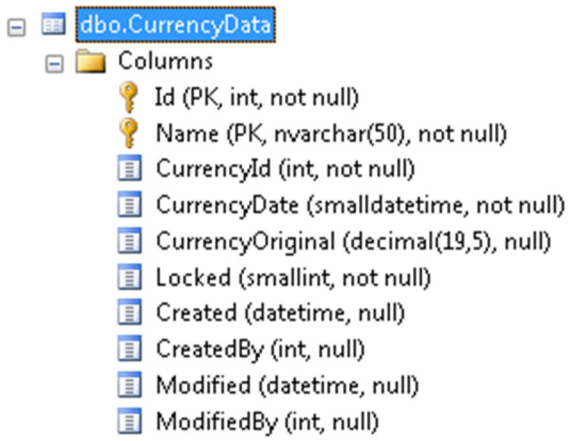
4.3 Tekninen toteutus

Tietokanta

Tietokantapalvelimena PlanMillin tuotantoympäristössä toimii Microsoftin MS SQL Server 2008 R2. Sen tietokannat on peilattu toiselle tietokantapalvelimelle. Jokaisella asiakasinstanssilla on oma tietokanta. Tietokannassa käytössä ovat taulut, näkymät, funktiot, proseduurit ja triggerit. Muita tietokantoja ovat vielä geneerinen aloituskanta, josta kopioidaan uudet tietokannat, ja vain luettava jaettu kanta, jossa kaikkien instanssien yhteiset parametrit ja kielitermit sijaitsevat. Projektin toteutus vaatii tietokantaan ainakin yhden uuden taulun, vanhojen taulujen muutoksia ja uusia funktioita ja proseduureja. Vanha tietokannan rakenne laskutuspuolelta löytyy liitteenä 2 olevasta ER-kaaviosta. Tietokannan valuuttakäsittelyssä tärkeintä roolia pitää CurrencyData-

taulu, joka ylläpitää jäädytettyä arvoa taulujen sarakkeiden valuutoista, jos ne eroavat instanssin oletusvaluutasta.

CurrencyData-taulu on dynaaminen minkä tahansa taulun sarakkeen valuuttatiedolle tarkoitettu talletuspaikka. CurrencyData-tauluun voi lisätä viittauksen taulun sarakkeeseen antamalla sille taulun rivin uniikin ID:n sarakkeeseen CurrencyData.Id ja taulun sekä sarakkeen nimen pisteellä eroteltuna sarakkeeseen CurrencyData.Name. Nämä kaksi saraketta, Id ja Name, rakentavat taulun primääriavaimen.



Kuva 5. CurrencyData-taulun sarakkeet, tietotyypit ja avaimet.

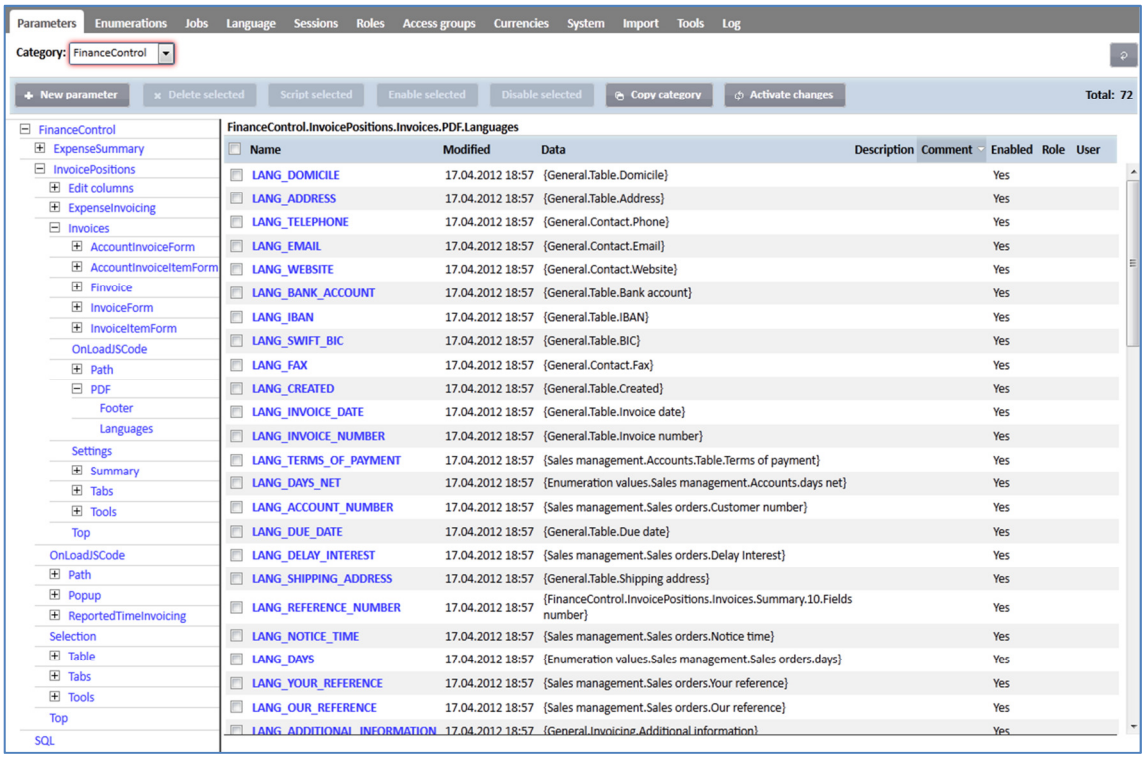
Tauluun talletetaan myös tieto valuutasta, päivämäärästä, alkuperäisestä summasta ja lukitustilasta. Muuta tietoa on luomis- ja muokkausajankohdat.

Parametrit

PlanMill-järjestelmä käyttää suuria määriä parametreja, joita on tällä hetkellä yli 30 000 kappaletta. Niillä kerrotaan kaikki käyttöoikeuksista moduulien rakenteisiin ja ennalta määriteltyihin SQL-lauseisiin (prepared statements). Parametrit ovat erittäin dynaaminen tapa muokata järjestelmää ja säätää sen asetuksia. Parametrimuutokset voidaan ottaa käyttöön yhdessä tai kaikissa instansseissa käytännössä reaaliaikaisesti, jolloin esimerkiksi asiakasinstanssien räätälöinti on todella helppoa milloin vain. Parametrit voidaan haluttaessa sitoa pelkästään käyttäjälle tai käyttäjäroolille antaen lisää joustavuutta räätälöintiin.

Instanssien yhteiset jaetut parametrit talletetaan Javan mustiin Tomcat-tasolle nopeaa hakua ja käyttöä varten. Näin jokainen instanssi käyttää jaettua muistiavaruutta yh-

teisten parametrien hakuun. Tämä alentaa muistin käyttöä, kun kuitenkin suurin osa parametreista on vakioparametreja, eikä niitä tarvitse jokaiselle instanssille erikseen monistaa. Jokaisella instanssilla on vielä oma muistiavaruus lokaaleille parametreille, joilla voidaan ylikirjoittaa jaetusta lähteestä saadut parametrit. Lisää parametrien lataamisesta muistiin löytyy esimerkkikoodista 5.

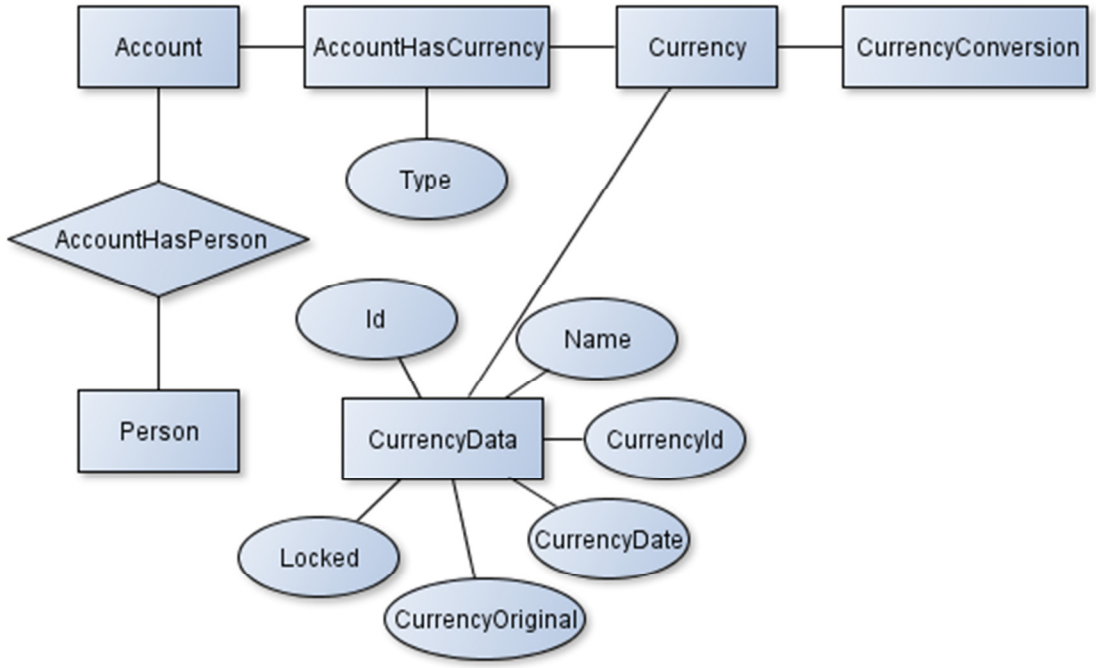


Kuva 6. Järjestelmän sisäinen parametri-editori. Käyttöliittymässä näkymään on valittuna uuden dynaamisen laskupohjan kieliparametrit.

Tietokannan muutokset

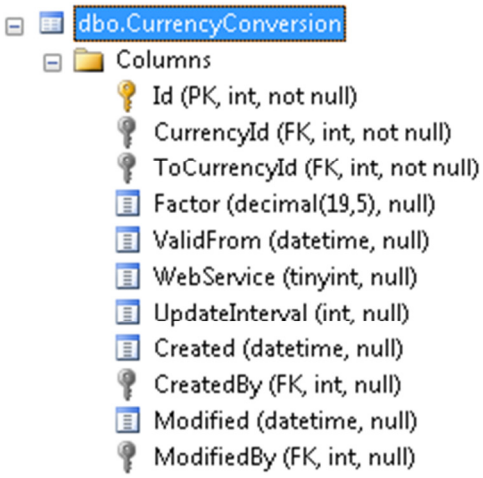
Tietokanta on aikaisemmin tukenut vain yhtä järjestelmänlaajuista oletusvaluuttaa, jota ei ole voinut enää muuttaa käyttöönoton jälkeen. Tämä on ongelmallista, jos yritys muuttaa tai jakautuu ulkomaille, jolloin myös olemassa oleva tieto pitäisi saada lokalisoitua kyseiseen maahan. Normaalien tietokannan taulujen valuutta-sarakkeet ovat yrityksen kirjanpitovaluutassa numeerisena. Jatkossa ainoa tapa tietää, mikä valuutta on kyseessä, on etsiä taulun riviin liittyvä yritys ja katsoa yrityksen voimassa oleva kirjanpitovaluutta. Tätä takaisinpäin linkitystä varten tehdään näkymä, johon liitetään kaikki tietokannan taulut ja rivit, jotka käsittelevät valuuttaa. Tietokannan näkymän generointilause taulun rivin ja yrityksen kirjanpitovaluutan väliselle yhteydelle löytyy liitteestä 3.

Tietokannan ainoa uusi taulu on AccountHasCurrency, joka sisältää relaatiotiedon yrityksestä ja siihen liittyvästä voimassaolevasta kirjanpitovaluutasta. Taululla on myös tieto valuutan tyypistä, koska yritykselle voidaan liittää jatkossa myös erikseen laskutusvaluuttoja.



Kuva 7. Uuden AccountHasCurrency-taulun sijoittuminen tietokantaan ER-kaaviossa.

Tästä voidaan muodostaa kaikki yritykseen liittyvät valuuttaparit CurrencyConversion-taululla, joka myös sisältää valuuttaparien voimassaolevat vaihtokurssit. CurrencyConversion-taulu sisältää vain yhden rivin per voimassaoleva valuuttapari. Valuuttapari määritellään kentissä CurrencyConversion.CurrencyId ja CurrencyConversion.ToCurrencyId.



Kuva 8. CurrencyConversion-taulun sarakkeet ja avaimet.

Muita tärkeitä kenttiä taulussa on

- Factor, eli vaihtokurssi
- ValidFrom, eli vaihtokurssin voimassaolon alkamisajankohdan
- WebService, eli onko automaattinen kurssinhaku päällä
- UpdateInterval, eli automaattisen kurssihaun päivitysfrekvenssi.

Käänteisten valuuttaparien ja niiden inversio-kurssien (käänteiset valuuttaparit) hakuun tullaan käyttämään indeksoitua virtuaalista näkymää PMVCurrencyPair, jonka sisältö talletetaan Java-sovelluksen muistiin nopeaa hakua varten. Tämä näkymä toteutetaan kuitenkin vasta seuraavaan versioon.

4.4 Laskentatarkkuus

Järjestelmässä kaikki valuuttakäsittely on tapahtunut aikaisemmin liukuluvuilla (floating point integer). Tämä toteutus saattaa aiheuttaa ongelmia varsinkin valuuttalaskennassa, koska liukuluvulla ei voi kuvata absoluuttisesti esimerkiksi arvoa 0,1 (IEEE 754-1985, 2012). Tämä on teknisesti mahdotonta, koska sille ei ole olemassa vastaavaa binääriarvoa, joista liukuluku syntyy. 2-lukujärjestelmässä 1/10 tuottaa tulokseksi loppumattoman murtoluvut 0,00011001100110011... riippumatta käytettyjen desimaalien määrästä. Liukuluvun sijasta tietokannassa tulisi käyttää tarpeeksi tarkkaa MS SQL:n

decimal-tietotyyppiä, esimerkiksi **DECIMAL(19, 5)**, jossa 19 on numeroiden kokonaismäärä ja 5 on desimaalien määrä kokonaisnumeromäärästä (Decimal and numeric (Transact-SQL), 2012). Tällä tarkkuudella yhden tietueen talletuskooksi tulee 9 tavua. Myös MySQL:ssä decimal-tietotyyppi määritellään DECIMAL(M, D).

Javassa on aivan sama ongelma liukulukujen tarkkuuden kanssa kuin kaikkialla muuallakin. Javassa tarvittava tarkkuus valuuttojen laskemiseen saadaan käyttämällä BigDecimal-olita, jossa desimaaliluvut tallentuvatkin kokonaislukuna ja desimaalipilkun paikka talletetaan erikseen. BigDecimal on immutable, eli muuttumaton, olio, jonka laskentametodit palauttavat uuden olion laskun lopputuloksen arvoilla. (BigDecimal, 2011; Paul, 2003.)

```
/*******************************
* BigDecimal on immutable olio. *
*******************************/

BigDecimal value1 = new BigDecimal(19);
BigDecimal result1 = value1.divide(100);
// result1: 19/100 = 0.19, integer=19, scale=2

BigDecimal value2 = new BigDecimal(21);
BigDecimal result2 = value2.divide(110);
// result2: 21/110 = 0.190, integer=190, scale=3

BigDecimal a1 = new BigDecimal(2);
BigDecimal b1 = new BigDecimal(3);
BigDecimal result3 = a1.divide(b1, 9, BigDecimal.ROUND_HALF_UP);
// result3: 0.666666667
```

Esimerkkikoodi 3. BigDecimal-luokan käyttö.

4.5 Lokalisoinnin toteutus

Lokalisointi kielten puolelta on toteutettuna enumeraatio-listana järjestelmän parametreihin. Vanhassa lokalisoinnissa parametrissa oli listattuna pelkästään kieli. Tähän parametriin ei kuitenkaan vielä tehdä muutoksia, koska se antaa mahdollisuuden vakiokielitermeille.

Taulukko 2. Vanhat lokalisointiasetukset parametrissa.

Parametri	**Sisältö**
Enumeration values.System.Language	en:{Enumeration values.Employee directory.Contact.English}; de:{Enumeration values.Employee directory.Contact.German}; fi:{Enumeration values.Employee directory.Contact.Finnish};

Järjestelmän termit sidotaan kuitenkin nyt kieleen ja maahan. Nykyisille termeille kielten maakoodi (tietokannassa LanguageCode) muutettiin oikeaksi. Esimerkiksi suomenkielisille termeille kielikoodiksi tuli fi-FI ja ruotsinkielisille sv-FI, jossa ensimmäinen osa kertoo kielen ja toinen osa kertoo maan. Ruotsinkielisiä termejä ei muuteta olemaan sv-SV, koska suurin osa termeistä on suomenruotsia.

Kielitermejä käytetään pääsääntöisestä järjestelmän parametreissa. Termi kääritään aaltosulkeisiin, jolloin järjestelmä osaa automaattisesti kääntää sen käyttäjän session tai muulle halutulle kielelle. Parametrin tietueessa esimerkin kielitermi esiintyy muodossa {General.Table.Invoice}. Alla on saman kielitermin muita käännöksiä eri kielille ennen kielikoodien muutoksia.

Taulukko 3. Esimerkkejä tietokannassa olevista termeistä ja käännöksistä vanhoilla lokalisaatioasetuksilla.

Id	Name	LanguageCode	Data
2699107	General.Table.Invoice	de	Rechnung
2806189	General.Table.Invoice	ee	Arve
2699103	General.Table.Invoice	en	Invoice
2699104	General.Table.Invoice	fi	Lasku
2806188	General.Table.Invoice	sv	Faktura
514522	General.Table.Invoice date	de	Faktura Datum
506076	General.Table.Invoice date	en	Invoice date
521606	General.Table.Invoice date	fi	Laskun pvm
504182	General.Table.Invoice number	de	Rechnungsnummer
501688	General.Table.Invoice number	en	Invoice number
521608	General.Table.Invoice number	fi	Laskun numero

Oikean kielitermin valinta tapahtuu Java-puolella parsimalla kielikoodi, eli kieli, maa ja variantti -yhdistelmä, osia ja vertaamalla niitä Javan muistissa oleviin kielikoodeihin ja termien nimiin. Kielitermit on tallennettu ParamControl-luokassa tietorakenteeseen **HashMap<String, Map<String, String>>**, jossa ylimmän tason kartan (HashMap) avaimena on kielikoodi ja sisemmässä kartassa (Map-rajapinta) avain-arvo-pareina on tietokannan kentät Language.Name ja Language.Data. Tietorakenteen populointi tapahtuu heti luokan alustuksessa static-osiossa, jossa tiedot ladataan erillisessä Loader-sisäluokassa.

```
public class ParamControl {
...
private static Loader container = new Loader();
...


/** Inner class for loading all data. */
    private static class Loader {
        public Map<String, Map<String, String>> langs =
            new HashMap<String, Map<String, String>>();
     ...

     /** (Constructor) */
     public Loader() {
         try {
             String sqlParams =
                 SQLMapping.getPreparedStatement(SQL_LOAD_PARAMS);
             loadParameters(sqlParams, null);
             String sqlLangs =
                 SQLMapping.getPreparedStatement(SQL_LOAD_LANGS);
             loadLanguages(sqlLangs, null);
             ready = true;
         } catch (Exception e) {
             Logger.log(e);
         }
     }
}
```

Esimerkkikoodi 4. Lähdekoodia parametrien ja kielitermien lataamisesta muistiin.

Jokaista kieli, maa ja variantti -kombinaatiota ei ladata muistiin vaan vasta hakuvaiheessa kyseltäessä kyseiseltä termiä tarkistetaan, löytyykö se jo muistista. Jos termiä ei löydy halutulla kielikoodilla, niin järjestelmä palauttaa rekursiivisesti korkeamman tason kombinaation. Esimerkiksi termille {General.Table.Invoice} ja kielikoodille en-GB-variant yritetään etsiä termikarttaa avaimella en-GB-variant. Jos karttaa ei löydy tai kartan palauttamaa termiä ei löydy, palaudutaan korkeammalle tasolle parsimalla taso kielikoodista, joka esimerkissä olisi vain en-GB, ja haetaan sillä avaimella termikarttaa. Tätä toistetaan, kunnes haluttu termi löytyy, esimerkissä viimeisellä avaimella en, tai

palautetaan tyhjä, jolloin ylemmän tason kutsuttava metodi palauttaa termin nimen aaltosulkeissa takaisin. Tämä mahdollistaa teoreettisen n-tason kielihierarkian, vaikkakin se ei ole tarpeellista todellisessa elämässä.

```
/** Gets an individual language text. */
private static String getLangText(String code, String name) {
    if (code == null) {
        return null;
    }

    Map<String, String> lang = container.langs.get(code);
    if (lang != null) {
        String value = lang.get(name);
        if (value != null) {
            return value;
        }
    }
    // recursive call
    return getLangText(getDefaultLangCode(code), name);
}

private static String getDefaultLangCode(String code) {
    int delimLoc = code.lastIndexOf(LANG_SEPARATOR);
    if (delimLoc == -1) {
        return null;
    } else {
        return code.substring(0, delimLoc);
    }
}
```

Esimerkkikoodi 5. Alimman tason sisäisesti käytettävä rekursiivinen hakumetodi kielitermin hakemiseen muistin tietorakenteesta.

Lokalisoinnissa tulemme myös lisäämään paremman tuen aluekohtaisille valuuttaformaateille. Valuutat on kyllä formatoitu jo entuudestaan, mutta koska valuutta on voinut olla vain yhdessä formaatissa, on se sama kaikille instanssin yrityksille riippumatta kielestä tai alueesta. Tämä ei toimi isomman konsernin sisällä, jossa käyttäjäyritykset käyttävät esimerkiksi erilaisia desimaalierottimia tai haluavat valuuttasymbolit eri puolelle valuuttaa. Aikataulun kireyden takia lokaalikohtainen valuuttaformatointi siirtyy myöhempään versioon.

4.6 Laskupohjat

Laskupohjat on PlanMillissä toteutettu XSL-tyylipohjina ja aikaisempien versioiden perintönä jokaiselle kieli ja valuutta -yhdistelmälle on ollut oma XSL-tyylipohja. Nämä tyylipohjat ovat kuitenkin käyttäneet laskupohjien yhteisiä aputyylipohjia, joissa esi-

merkiksi sisään tulevat parametrit on määritelty. Tämä toteutus on ollut vaikea ylläpitää ja tarvittaessa räätälöidä asiakkaan tarpeisiin. Ennen muutosta laskujen tyylipohjia on ollut tuotannossa noin reilut 100 kappaletta, kun huomioidaan vakiopohjat, Finvoice-verkkolaskupohjat ja kaikkien asiakkaiden räätälöidyt tyylipohjat. Laskupohjien nimet ovat olleet seuraavaa muotoa.

```
planmill_fi_eur.xsl
planmill_fi_eur_creditnote.xsl
planmill_en_eur.xsl
planmill_en_eur_creditnote.xsl
planmill_en_gbp.xsl
planmill_en_gbp_creditnote.xsl
```

Esimerkkikoodi 6. Otteita laskujen tyylipohjista ennen muutoksia.

XSL-tyylipohjat toimivat Javassa Xalan-Java-paketilla, joka on saatavilla Apachen verkkosivuilta (Xalan-Java, 2006). XML-dokumentin ajaminen XSL-tyylipohjan läpi tuottaa XSL-FO-tiedoston, joka ajamalla Apache FOP:in (FOP, 2012) läpi tuottaa lopputulokseksi HTML-, XML- tai PDF-sivuja.

Projektissa päätettiin suunnitella tyylipohjat uudestaan siten, että tyylipohjia olisi vain kaksi yleisimmin käytössä ollutta versiota. Kaikki kielet, termit ja symbolit tulevat pohjille parametreina ja näitä parametreja voi muokata järjestelmässä reaaliajassa, eikä tyylipohjille tarvitsisi tehdä enää muutoksia. Poikkeuksena on tietysti asiakkaille räätälöidyt erikoisemmat pohjat, mutta nekin olisi helpommin ylläpidettävissä parametroinnilla.

Jatkossa tulee olemaan vain seuraavat laskutyylipohjat:

```
…/invoicetemplates/custom/...
…/invoicetemplates/generic/languages.xsl
…/invoicetemplates/generic/parameters.xsl
…/invoicetemplates/generic/functions.xsl
…/invoicetemplates/generic/layout.xsl
…/invoicetemplates/generic/header.xsl
…/invoicetemplates/generic/body.xsl
…/invoicetemplates/generic/footer.xsl
…/invoicetemplates/template.xsl
```

Esimerkkikoodi 7. Laskupohjien tiedostot.

Hakemistoista custom/ sisältää asiakkaiden räätälöidyt tyylipohjat ja generic/ yleisesti käytetyt tyylipohjat kuten muuttujat, funktiot ja sommittelun. Template.xsl on vain dynaaminen pohja, jota voi käyttää millä tahansa kielellä ja valuutalla. Tämä pohja tukee myös hyvityslaskuja.

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsl:stylesheet version="1.0"
          xmlns:xsl=http://www.w3.org/1999/XSL/Transform
          xmlns:fo="http://www.w3.org/1999/XSL/Format">

      <xsl:include href="generic/parameters.xsl" />
      <xsl:include href="generic/functions.xsl" />
      <xsl:include href="generic/languages.xsl" />
      <xsl:include href="generic/layout.xsl" />
      <xsl:include href="generic/header.xsl" />
      <xsl:include href="generic/footer.xsl" />
      <xsl:include href="generic/body.xsl" />

      <xsl:template match="/">
          <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
              <xsl:call-template name="layout" />
              <fo:page-sequence
                       initial-page-number="1"
                       master-reference="PageMaster">
                  <xsl:call-template name="header" />
                  <xsl:call-template name="footer" />
                  <xsl:call-template name="body" />
              </fo:page-sequence>
          </fo:root>
      </xsl:template>
  </xsl:stylesheet>
```

Esimerkkikoodi 8. Template.xsl:n sisältö. Pohja vain koostaa muut pohjat.

Kielitermien vienti tyylipohjaan tapahtuu parametreilla. Tässä lyhyt otos generic/languages.xsl:n sisällöstä:

```
<xsl:param name="LANG_VAT" />
<xsl:param name="LANG_PAGE" />
<xsl:param name="LANG_COMPANY" />
<xsl:param name="LANG_BUSINESS_ID" />
<xsl:param name="LANG_INVOICE_DATE" />
```

Esimerkkikoodi 9. Parametrien tuonti tyylipohjalle.

Seuraavaksi esimerkki LANG_INVOICE_DATE-parametrin käytöstä body.xsl-tyylipohjassa. Itse kentän arvon formatointi tapahtuu jo Java-puolella samojen lokaaliasetusten perusteella kuin kielitermien käännökset.

```
<fo:table-cell
        padding="2pt"
        border-left-style="solid"
        border-left-width="0.3mm"
        border-right-style="solid"
        border-right-width="0.3mm"
        border-top-style="solid"
        border-top-width="0.3mm">
    <fo:block font-size="8pt" line-height="12pt"
            wrap-option="wrap">
        <fo:block font-weight="bold">
            <xsl:value-of select="$LANG_INVOICE_DATE"/>
        </fo:block>
        <xsl:value-of select="/Invoice/Header/Date"/>
    </fo:block>
</fo:table-cell>
```

Esimerkkikoodi 10. LANG_INVOICE_DATE-parametrin käyttö body.xsl-tyylipohjassa.

Finvoice

PlanMill-järjestelmä osaa tehdä myös Finvoice-verkkolaskuja. Finvoice on yksi Suomessa käytetyistä verkkolaskun sanomamuodoista. Verkkolasku on sähköinen lasku, jonka tiedot välitetään tyypillisesti XML-sanomana. Verkkolasku sisältää kaikki tarvittavat tiedot laskun maksamiseen, mutta usein se voi sisältää myös lisää tietoa, kuten laskuerittelyn tehdyistä tunneista ja mahdollisista kuluista.

Finvoice-laskujen muodostus toimii PlanMillissä käytännössä samalla tavalla kuin normaalillakin laskulla, mutta se on parametroitu eri tavalla, käyttää eri pohjaa ja lopputuloksena syntyy PDF:n sijasta XML-dokumentti.

Finvoice-laskujen luonnissa on ollut sama ongelma staattisen laskupohjaan sidotun lokalisoinnin kanssa kuin normaaleilla laskuilla. Tyylipohjat ovat aikaisemmin olleet muotoa:

```
…/invoicetemplates/finvoice_fi_eur.xsl
…/invoicetemplates/finvoice_fi_eur_creditnote.xsl
…/invoicetemplates/finvoice_en_eur.xsl
…/invoicetemplates/finvoice_en_eur_creditnote.xsl
…/invoicetemplates/finvoice_en_gbp.xsl
…/invoicetemplates/finvoice_en_gbp_creditnote.xsl
```

Esimerkkikoodi 11. Finvoice-tyylipohjat ennen muutoksia.

Tämä on epäkäytännöllinen tapa, kuten jo normaaleista laskupohjista todettiin, joten myös ne muutettiin yhdeksi dynaamiseksi XSL-pohjaksi nimeltään finvoicetemplate.xsl. Jatkossa tiedostorakenne on seuraavanlainen:

```
…/invoicetemplates/Finvoice.dtd
…/invoicetemplates/Finvoice.xsd
…/invoicetemplates/Finvoice.xsl
…/invoicetemplates/finvoicetemplate.xsl
```

Esimerkkikoodi 12. Finvoice-tiedostot hakemistorakenteessa.

Hakemistossa olevat kolme muuta tiedostoa, Finvoice.dtd, Finvoice.xsd ja Finvoice.xsl, ovat Finvoice-standardin mukaisia validointitiedostoja, eikä niihin tehdä mitään muutoksia. Finvoicetemplate.xsl saa kielitermit ja esimerkiksi tiedon, onko se hyvityslasku, parametreina, jotta formatointi toimisi dynaamisesti.

4.7 Testaus

Tiesimme, että BDD-tarinatiedostoja tulee olemaan paljon ja niitä tulevat kehittämään useat konsultit, joten ylläpidossa tulee olemaan ongelmia. Ratkaisimme asian siirtämällä tiedostot PlanMillin wikiin, jossa ne ovat reaaliaikaisesti kaikkien saatavilla, muutoshistoria on tiedossa ja tekstin tyylittely helppoa. Tyylitelty esimerkki BDD-tarinasta löytyy kuvasta 3 s. 16.

Testien ajo tapahtuu tällä hetkellä lokaalina. Tämän takia jouduin ohjelmoimaan pienen lisäohjelman ajurille, joka osaa ladata testit suoraan wikistä testiympäristöön. Ajurille annetaan wiki-sivun sisäinen id, ja ajuriin lisätty tiedostolataaja osaa siten ladata oikean testin koneelle. Koska testit ovat myös riippuvaisia toisistaan, osaa lataaja rekursiivisesti ladata myös kaikki muutkin testit koneelle. Kun lataukset ovat valmiita ajettaviksi, lähtee ajuri suorittamaan riippuvuushierarkiassa alimpaa eli ensimmäistä testiä. Valitettavasti tällä hetkellä jBehave-ajuri ei osaa sisäisesti selviytyä tilanteesta, jossa testien riippuvuudet toisiinsa aiheuttavat loppumattoman syklin.

Testiajuri ohjelmoitiin tulkkaamaan yleisimpiä BDD-tarinoiden käskyjä, eli suorittamaan halutun selaintoiminnallisuuden Seleniumilla. Ajuri käyttää Steps-luokkia, joissa tarinakomennot yhdistetään Selenium-kutsuun.

Kuvan 3 esittämä ensimmäinen skenaario tulkataan jBehaven LoginSteps-luokassa seuraavalla metodilla:

```
@Given("{I|you} {successfully|unsuccessfully|} used address " +
           "$planmillinstance, username $username, password " +
           "$password and clicked $loginButton and {I|you} see " +
           "$firstname $lastname in PlanMill home page")
@Composite(steps = {
           "Given I open the browser and go to $planmillinstance",
           "When I sign in with username $userId",
           "When I enter password $password",
           "When I click login page button $loginButton",
           "Then check I am logged in as $firstname $lastname"})
public void doSignInWithDefaultAuthentication(
           @Named("planmillinstance") String planmill_instance,
           @Named("userId") String userId,
           @Named("password") String password,
           @Named("loginButton") String loginButton,
           @Named("firstname") String user_first_name,
           @Named("lastname") String user_last_name) {}
```

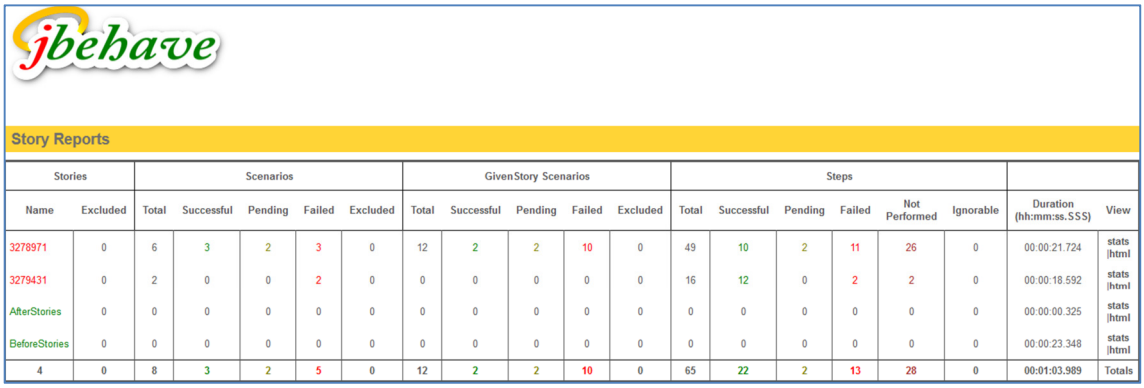
Esimerkkikoodi 13. Kuvion 3 esittämää skenaariota tulkkaava Java-metodi.

Annotaatiossa @Composite kerrotaan järjestys, jossa spesifimmät askeleet (steps) tullaan suorittamaan. Esimerkkinä ensimmäisen askeleen tulkkaava metodi:

```
@Given("{I|you} open the browser and go to $planmillinstance")
@Aliases(values = {
    "{I|you} open the browser and go to <planmillinstance>",
    "{I|you} open PlanMill by going to address $planmillinstance"})
public void openBrowserAndGoToInstance(
        @Named("planmillinstance") String planmill_instance) {
    loginPage
        .manage()
        .timeouts()
        .implicitlyWait(3, TimeUnit.SECONDS);
    loginPage.get(planmill_instance);
}
```

Esimerkkikoodi 14. Edellisen koodiesimerkin ensimmäisen @Composite-annotaation vastaava metodi.

Testien tulos voidaan automaattisesti viedä verkkosivulle, jossa on luettavissa kaikkien testien osien loki-tiedostot, mahdolliset virheet ja kuvakaappaus käyttöliittymästä virhetilanteessa. Kuvassa 9 on testien loppuraportti.



jbehave

Story Reports

Stories		Scenarios					GivenStory Scenarios					Steps							
Name	Excluded	Total	Successful	Pending	Failed	Excluded	Total	Successful	Pending	Failed	Excluded	Total	Successful	Pending	Failed	Not Performed	Ignorable	Duration (hh:mm:ss.SSS)	View
3278971	0	6	3	2	3	0	12	2	2	10	0	49	10	2	11	26	0	00:00:21.724	stats |html
3279431	0	2	0	0	2	0	0	0	0	0	0	16	12	0	2	2	0	00:00:18.592	stats |html
AfterStories	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00:00:00.325	stats |html
BeforeStories	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00:00:23.348	stats |html
4	0	8	3	2	5	0	12	2	2	10	0	65	22	2	13	28	0	00:01:03.989	Totals

Kuva 9. jBehave-ajurin loppuraportti testeistä. Kuvassa näkyy kahden testin statistiikkaa.

Selainajuritestaus on hidasta, eikä sitä voi suorittaa jokaisen pienen kehitysaskeleen jälkeen. Täten sitä tulisi käyttää vain isompien kokonaisuuksien, vaikka lomakkeiden, hyväksyntätestaukseen.

5 Lopputulos

Projekti lähti hyvin käyntiin, ja pääsimme nopeasti toteuttamaan tietokantamuutoksia ja käyttöliittymää. Projektipäällikkö kuitenkin päätti, että nyt olisi sopiva aika ruveta käyttämään yritykselle uutta BDD-testausmenetelmää, jolla voitaisiin myös tehdä suurin osa määrittelyistä. BDD-testauksen käyttöönotto ei sujunutkaan ihan niin jouhevasti kuin aluksi kuviteltiin, ja projektin resurssit jouduttiin siirtämään testauksen kehitykseen ja käyttöönottoon. Kun mukaan tuli vielä tarve monimutkaisen valuuttaparigenerointilogiikan yksikkötestaukselle TDD:nä, niin itse projektin eteneminen pysähtyi kokonaan.

Kun uusien testaus- ja kehitysvälineiden ja prosessien käyttöönotto oli saatu pidemmälle ja pakolliset asiakasprojektit ja muut projektia viivästyttäneet asiat hoidettua, jäljellä oli noin 8 viikon aikaikkuna, jossa projektista merkittävä osa piti saada tuotantoon. Koska muutoksia oli tulossa paljon järjestelmän ydintoimintoihin ja räätälöinnit piti ottaa huomioon, päädyttiin siihen, että muutoksia lähdettiin hakemaan kevyimmin muutettavasta päästä, joka kuitenkin toisi merkittäviä parannuksia sekä asiakkaiden kokemaan palveluun että sisäiseen tehokkuuteen.

Lisäksi todettiin, että ajanpuutteen vuoksi ensimmäisessä vaiheessa ei pystyttäisi järkevästi ottamaan käyttöön usean kirjanpitovaluutan tukea samassa ympäristössä, eikä

suurin osa asiakaskunnasta sitä edes tarvitse, joten pääpaino oli tukea yhden kirjanpitovaluutan ja monen laskutusvaluutan ja -kielen yhdistelmää mahdollisimman hyvin ensimmäisessä vaiheessa. Esimerkiksi laskupohjien ylläpito nykyisellä mallilla aiheutti herkästi asiakkailta tilauksen maksullisesta räätälöinnistä ja kohtalaisen hitaan ja manuaalista työtä vaativan palveluprosessin, jossa pahimmillaan räätälöitiin sama asia usealle asiakkaalle moneen kertaan.

Versiopäivitys

Kevyet muutokset saimme tehtyä valmiiksi PlanMillin version 14.4 tuotantopäivitykseen mennessä, joka oli 23. toukokuuta 2012. Nämä muutokset sisälsivät työssä esitellyt laskupohjamuutokset ja kielitermien lokalisointituen. Muita muutoksia järjestelmään tuli myyntitilaus- ja laskusivuille, joissa käyttöliittymässä näytetään nyt valuuttamäärät myyntitilauksen tai laskun valuutassa. Aikaisemmin käyttöliittymän kaikki valuuttamäärät näytettiin käyttäjän asetusten valuuttana, paitsi projektisivuilla, missä voi projektikohtaisesti valita projektin valuutan.

Versiopäivitys meni kohtuullisen jouhevasti ja kaikki raskaasti räätälöityjen asiakasinstanssien laskupohjat saatiin testattua. Kaikki laskupohjaräätälöinnit piti muuttaa vastaamaan uusia laskupohjia ja tässä onnistuttiin kohtuullisen hyvin. Päivityksessä kuitenkin ilmeni ongelma vanhojen räätälöityjen laskupohjien riippuvuuksissa vanhoihin vakiopohjiin ja nämä jouduttiin jättämään tuotantoympäristöön, kunnes räätälöidyt pohjat saadaan vakioitua tai siirrettyä kaikki riippuvuudet räätälöidyn pohjan sisään. Tilanne on tällä hetkellä vielä aika sekava, kun tuotantoympäristössä on kolmen sukupolven laskupohjia; uudet dynaamiset pohjat, vanhat vakiopohjat ja asiakkaille räätälöidyt pohjat. Kaikki kuitenkin toimii ja seuraavaan versiopäivitykseen mennessä olisi tarkoitus päästä järkevään tilanteeseen pohjien kanssa.

Suuremmat ja kriittisemmät muutokset käyttäjäyrityksen hallintaan ja tuki usealle oletusvaluutalle pidetään vielä kehitysympäristössä jatkokehitystä ja testausta varten. Projektien osien pilkkomisesta ja priorisoinnista johtuen nämä muutokset voidaan toteuttaa aikaisintaan tämän vuoden viimeisimmällä neljänneksellä. Tähän lykkäykseen vaikuttaa myös tuleva kesäkausi ja työntekijöiden kesälomat. Samalla saamme lisää

aikaa rauhassa ratkaista haasteet räätälöidyissä laskupohjissa ja niiden yhteistoiminnan uusien vakiopohjien kanssa.

Kehitysideat

Uskon, että seuraavalla kerralla isompi järjestelmän kehitysprojekti testauksen kanssa on vakioidumpaa eikä sen kehittämiseen ja opetteluun mene enää niin suurta osuutta ajasta. Myös uusien kehitystyökalujen käytössä osaaminen on koko organisaatiossa noussut jo tämän projektin aikana ja siltäkin osin seuraavissa projekteissa työskentely tulee olemaan nopeampaa. Oikein käytettynä uudet työkalut helpottavat ja yksinkertaistavat sovelluskehitystä ennen niin herkässä ekoympäristössä.

Tulevissa projektissa tulisi tehdä enemmän selviä päätöksiä linjavedoista ja tarkempia aikatauluja projektin eri osien toteuttamiseen. Projekti on venynyt nykyisten toimintojen toteutustavan dokumentoimattomuuden, ohi ajaneiden asiakasprojektien, klusterointiprojektin venymisen, projektiryhmän sisäisten kommunikaatio- ja opiskeluhaasteiden sekä yritystä vaivanneiden sisäilmaongelmien aiheuttamien sairaspoissaolojen vuoksi. Lähtötilanteessa tiedettiin, että projekti on haastava ja sisältää paljon asioita, joihin ei ole olemassa ”oikeaa vastausta”, ja asioita, joita täytyy tutkia ja varmistaa. Ketterän kehitysprosessin ja projektiin sisältyneen epävarmuuden vuoksi tarkkaa projektisuunnitelmaa ei edes yritetty tehdä koko projektin ajalle muutoin kuin hyvin karkealla tasolla. Koko kehitysprosessin eläminen on vaikeuttanut keskittymistä projektin kehitykseen ja venyttänyt sitä entisestään. Jatkossa toivoisin pystyväni omalta osaltani pitämään projektit kontekstissaan ja tehdä valmista toiminnallisuutta osissa, joista on helppo rakentaa kokonaista toiminnallisuutta.

Lokalisoinnissa ei keritty tekemään kuin pintaraapaisu. Muuta lokalisoitavaa järjestelmässä voisivat olla esimerkiksi kapasiteettikalenterit, jotka pitäisi liittää tiettyyn maahan. Yritysten poissaolokäytännöt ovat vähintään maakohtaisia ja useimmiten myös toimialakohtaisia. Jatkossa voidaan automatisoida lokalisoitu kapasiteettikalenteri vaikuttamaan suoraan juhlapyhiin ja työpäivän kestoihin. Myös verokannat vaihtelevat paljon eri maiden ja mahdollisesti myös kaupunkien välillä.

Lähteet

Acceptance testing. (10.4.2012). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Acceptance_testing>. Luettu 18.4.2012.

BCP 47. (syyskuu 2009). Verkkodokumentti. The Internet Engineering Task Force (IETF). <http://tools.ietf.org/html/bcp47>. Luettu 22.4.2012.

Behavior Driven Development. (16.4.2012). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Behavior_Driven_Development>. Luettu 17.4.2012.

BigDecimal. (2011). Verkkodokumentti. Oracle Java 6 API. <http://docs.oracle.com/javase/6/docs/api/java/math/BigDecimal.html>. Luettu 17.4.2012.

Concept of Currency Cross Triangulation. (26.4.2010). Verkkodokumentti. NobleTrading. <http://blog.nobletrading.com/2010/04/concept-of-currency-cross-triangulation.html>. Luettu 23.4.2012.

Currency Converter. (18.4.2012). Verkkodokumentti. OANDA. <http://www.oanda.com/currency/converter>. Luettu 18.4.2012.

Currency pair. (24.2.2012). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Currency_pair>. Luettu 18.4.2012.

Decimal and numeric (Transact-SQL). (2012). Verkkodokumentti. Microsoft Developer Network (Sql Server). <http://msdn.microsoft.com/en-us/library/ms187746.aspx>. Luettu 22.4.2012.

ECB Euro FX. (20.4.2012). Verkkodokumentti. ECB. <http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml>. Luettu 20.4.2012.

FOP. (2012). Verkkodokumentti. Apache. <http://xmlgraphics.apache.org/fop/>. Luettu 22.4.2012.

Forex Basics. (2012). Verkkodokumentti. GoForex. <http://www.goforex.net/forex-basics.htm>. Luettu 18.4.2012.

Hellesøy, A. (2012). Cucumber - Making BDD fun. Verkkodokumentti. Cucumber. <http://cukes.info/>. Luettu 17.4.2012.

IEEE 754-1985. (17.4.2012). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/IEEE_754-1985>. Luettu 17.4.2012.

Interbank foreign exchange market. (19.3.2012). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Interbank_foreign_exchange_market>. Luettu 18.4.2012.

Internationalization Best Practices: Specifying Language in XHTML & HTML Content. (12.4.2007). Verkkodokumentti. W3C. <http://www.w3.org/TR/i18n-html-tech-lang/#ri20030510.102829377>. Luettu 22.4.2012.

ISO 4217. (15.4.2012). Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/ISO_4217>. Luettu 20.4.2012.

jBehave. (12. huhtikuuta 2012). Verkkodokumentti. jBehave. <http://jbehave.org/>. Luettu 18.4.2012.

Live Exchange Rates. (18.4.2012). Verkkodokumentti. OANDA. <http://www.oanda.com/currency/live-exchange-rates/>. Luettu 18.4.2012.

Locale. 2004. Verkkodokumentti. The Open Group Base Specifications. <http://pubs.opengroup.org/onlinepubs/007904875/basedefs/xbd_chap07.html>. Luettu 22.4.2012.

Localization vs. Internationalization. (5.12.2005). Verkkodokumentti. W3C. <http://www.w3.org/International/questions/qa-i18n>. Luettu 22.4.2012.

Melik, R. (2002). Professional Services Automation. New York: John Wiley & Sons, Inc.

Paul, T. (heinäkuuta 2003). Working with Money in Java. Verkkodokumentti. Java ranch. <http://www.javaranch.com/journal/2003/07/MoneyInJava.html>. Luettu 22.4.2012.

PSA-järjestelmä. (13.10.2011). Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/PSA-järjestelmä>. Luettu 19.4.2012.

Scrum. (7. helmikuuta 2012). Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Scrum>. Luettu 18.4.2012.

Toiminnanohjausjärjestelmä. (26.12.2011). Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Toiminnanohjausjärjestelmä>. Luettu 28.3.2012.

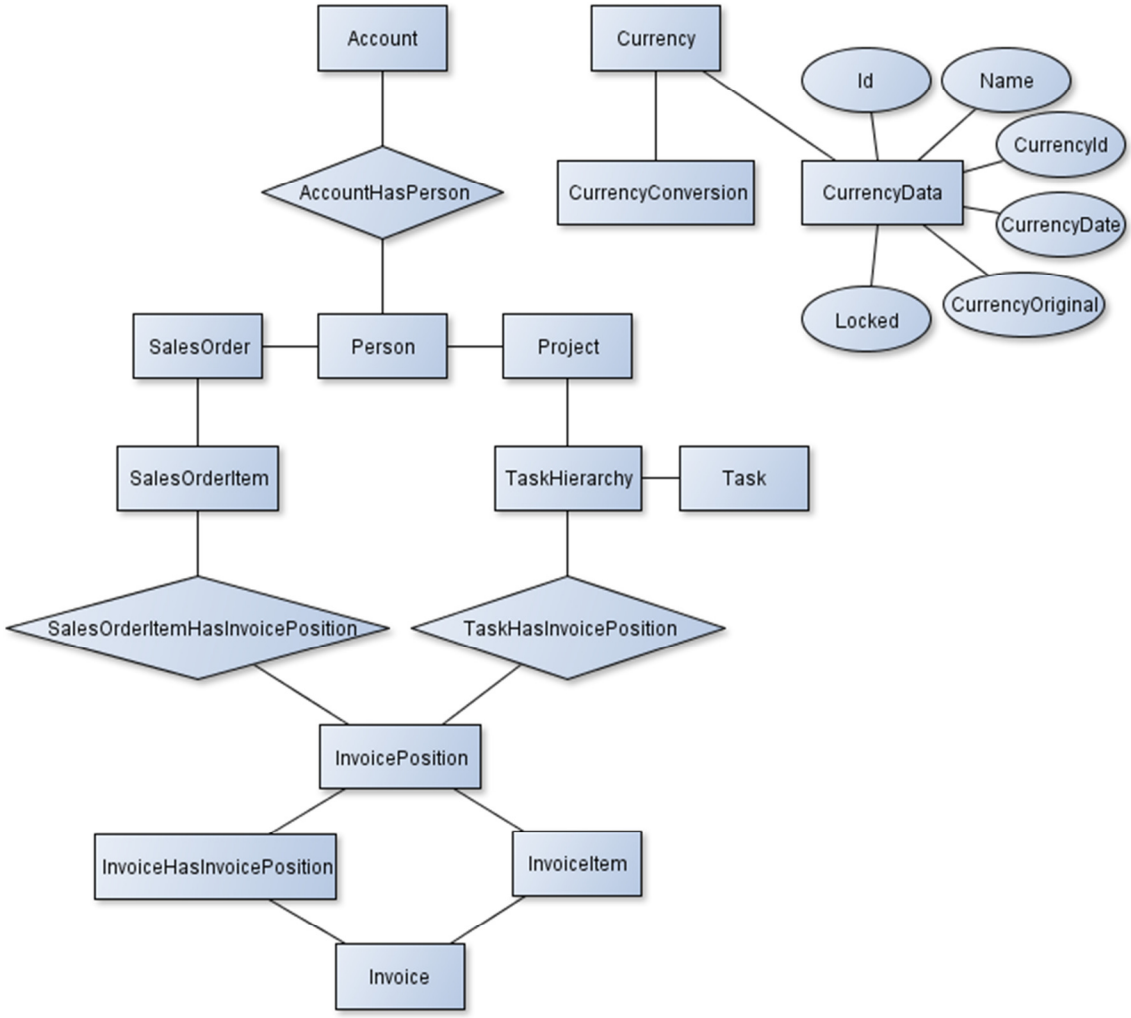
Valuutta. (5.4.2012). Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Valuutta>. Luettu 23.4.2012.

Xalan-Java. (2006). Verkkodokumentti. Apache XML. <http://xml.apache.org/xalan-j/>. Luettu 20.4.2012.

PlanMill POM (Project Object Model), typistetty versio

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
            http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.planmill</groupId>
    <artifactId>planmill</artifactId>
    <packaging>jar</packaging>
    <version>14.4</version>
    <name>planmill</name>
    <url>http://maven.apache.org</url>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-eclipse-plugin</artifactId>
                <version>2.9</version>
                <configuration>
                    <downloadSources>true</downloadSources>
                    <downloadJavadocs>true</downloadJavadocs>
                    <sourceExcludes>
                        <sourceExclude>**/.svn/**</sourceExclude>
                    </sourceExcludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <repositories>
        <repository>
            <id>spy</id>
            <name>Spy Repository</name>
            <layout>default</layout>
            <url>http://files.couchbase.com/maven2/</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
    <dependencies>
        <dependency>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-servlet-api</artifactId>
            <version>7.0.26</version>
        </dependency>
        <dependency>
            <groupId>com.hazelcast</groupId>
            <artifactId>hazelcast</artifactId>
            <version>1.9.3.4</version>
        </dependency>
    ...
    </dependencies>
</project>
```

ER-kaavio vanhasta tietokannan rakenteesta liittyen laskutukseen



PMVAccountCurrencyObjects-näkymän luontilause

```
CREATE VIEW [dbo].[PMVAccountCurrencyObjects] WITH SCHEMABINDING
    AS
    SELECT
        AccountHasCurrency.AccountId as 'AccountId',
        Opportunity.Id as 'ObjectId',
        'Opportunity.Id' as 'ObjectName',
        AccountHasCurrency.CurrencyId as 'CurrencyId'
    FROM
        Opportunity
        INNER JOIN AccountHasPerson ON
            Opportunity.ResponsibleId = AccountHasPerson.PersonId
        INNER JOIN AccountHasCurrency ON
            AccountHasPerson.AccountId = AccountHasCurrency.AccountId
                AND AccountHasCurrency.Type = 1
    UNION
    SELECT
        AccountHasCurrency.AccountId,
        SalesOrder.Id,
        'SalesOrder.Id',
        AccountHasCurrency.CurrencyId
    FROM
        SalesOrder
        INNER JOIN AccountHasPerson ON
            SalesOrder.ResponsibleId = AccountHasPerson.PersonId
        INNER JOIN AccountHasCurrency ON
            AccountHasCurrency.AccountId = AccountHasPerson.AccountId
                AND AccountHasCurrency.Type = 1
    UNION
        ...
    GO;
```